UQAC
Université du Québec
à Chicoutimi

**THÈSE**

PRÉSENTÉE À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

COMME EXIGENCE PARTIELLE

DU DOCTORAT EN INFORMATIQUE

**PAR**

IMÈNE BENKALAI

# ORDONNANCEMENT D'ATELIERS EN PRÉSENCE D'OPÉRATEURS

NOVEMBRE 2018

## TABLE DES MATIÈRES

## TABLE DES FIGURES

## LISTE DES TABLEAUX

## RÉSUMÉ

La théorie de l'ordonnancement a, depuis son avènement, suscité un grand intérêt de la part de chercheurs, de scientifiques, mais aussi d'industriels. Ceci est dû à la grande variété de problèmes réels pouvant être modélisés sous forme de problèmes d'ordonnancement. En effet, ce domaine peut trouver des applications aussi bien en gestion d'horaires qu'en informatique, ou encore en environnement de production.

Les études relativement récentes dans le domaine ont vu l'introduction du paramètre humain et sa considération dans la prise de décision portant sur les ressources matérielles d'un problème d'ordonnancement.

La présente thèse traite des problèmes d'ordonnancement d'ateliers avec opérateurs. Dans lesdits ateliers, des tâches devront être exécutées par plusieurs machines selon un ordre qui dépend du type d'atelier. Pour ce faire, lesdites tâches utilisent simultanément un opérateur et une machine. Le nombre d'opérateurs ainsi que leurs placements, *i.e.* leurs affectations aux machines, dépendra du modèle d'affectation choisi.

Tout d'abord, nous considérons un problème de flow shop de permutation avec temps de réglages. Nous supposons que le nombre d'opérateurs est égal au nombre de machines et qu'ils

s'occupent des opérations de réglage. Nous utilisons pour la résolution la métaheuristique Migrating Birds Optimization. Nous apportons des améliorations à l'algorithme de base et en présentons quatre versions, ce qui nous permet d'obtenir des résultats de relativement bonne qualité avec des configurations différentes qui apportent de la flexibilité lors de la prise de décision.

Par la suite, nous étudions des problèmes où le nombre d'opérateurs est inférieur au nombre de machines. Nous étudions trois types d'ateliers : les flow shops, les job shops et les open shops. Nous commençerons d'abord par l'étude de complexité de nos problèmes. Nous présentons d'abord des cas résolubles en temps polynomial et exhibons les méthodes permettant de les résoudre. Pour les cas difficiles, nous proposons des méthodes de résolution ainsi que des bornes inférieures. Les résultats montrent que les méthodes proposées donnent de bons résultats, souvent proches des bornes théoriques.

## ABSTRACT

Since its advent, scheduling theory has generated great interest amidst researchers, scientists but also industrialists. This is due to the great diversity of real problems that can be modeled as scheduling problems. Indeed, this field can find applications in timetabling, computer science but also in production systems.

Recent studies in the field have introduced the human resources and considered them in decision making processes involving the material resources of scheduling problems.

This thesis deals with scheduling shop problems with operators. In the aforementioned shops, tasks are to be processed according to orderings that depend on the type of shop. In order to do so, the tasks need simultaneously an operator and a machine. The number of operators and their positions in the shop, *i.e.* their assignements to machines, depends on the chosen assignment mode.

First, we consider a permutation flow shop problem with setup times. We assume that the number of operators is equal to the number of machines and that they handle setup operations. We use the metaheuristic called the Migrating Birds Optimization to solve this problem. We improve the basic algorithm and present four versions, which allows us to obtain results of

good quality with different structures, which provides flexibility in decision making.

Next, we study problems where the number of operators is less than the number of machines. We study three types of shops : flow shops, job shops and open shops. We first start by studying the complexity of our problems. Then we present well-solvable cases as well as their solution methods. For some $\mathscr{NP}$-hard cases, we propose solution methods and a lower bound. The results show that the proposed methods provide good results, often close to the theoretical lower bounds.

## REMERCIEMENTS

L'aboutissement de cette thèse n'aurait pas été possible sans le soutien et les encouragements de certaines personnes. Je sais d'avance que mes mots ne sauront pas leur rendre justice pour tout ce qu'ils ont pu m'apporter.

Je tiens à remercier avant tout mon directeur, Pr. Djamal Rebaine qui m'a offert la fabuleuse opportunité de rejoindre l'équipe du Groupe de Recherche en Informatique à l'Université du Québec à Chicoutimi. Mon expérience en tant que son étudiante fût des plus enrichissantes, j'ai appris bien plus que je n'avais osé l'espérer. Je le remercie pour son soutien constant, sa patience, sa compréhension, ses conseils et son dévouement.

Je tiens aussi à remercier mon co-directeur, Pr. Pierre Baptiste de l'École Polytechnique de Montréal. Ses conseils m'ont été précieux et m'ont beaucoup aidée à avancer, il a aussi su être patient à mon égard et apporter une touche d'humour à nos échanges, ce qui a contribué à me faire aimer encore plus ce que je fais.

Je tiens aussi à les remercier pour leur aide financière qui m'a permis de me concentrer sur mes recherches ainsi que pour leur rigueur lors des nombreuses relectures de la thèse et des papiers scientifiques.

J’exprime aussi mes plus sincères remerciements à l’équipe du Département d’Informatique et Mathématique pour leur constante bonne humeur qui permet d’offrir un accompagnement agréable à tous ses étudiants.

Enfin, je ne peux clore cette section sans remercier ceux qui ont été là pour moi depuis toujours, ma famille. À ma mère, mon père et mon petit frère : merci d’exister.

Je remercie aussi mon mari pour son soutien inconditionnel et je conclut avec un merci spécial pour ceux qui ont été ma famille à Chicoutimi.

# INTRODUCTION GÉNÉRALE

La théorie de l'ordonnancement compte parmi les disciplines les plus étudiées en recherche opérationnelle. Sa grande popularité est due au fait que, non seulement elle permet de modéliser un grand nombre de problèmes pratiques dans des domaines aussi importants que variés, mais elle est également importante sur le plan théorique où elle permet la conception de divers modèles sur lesquels peuvent s'appuyer des études futures.

Dans un problème d'ordonnancement classique, il s'agira d'ordonnancer un ensemble de $n$ tâches $J = \{1, 2, \ldots, n\}$ devant être réalisées par un ensemble de $m$ machines $M = \{M_1, \ldots, M_m\}$. La manière d'ordonnancer dépendra par la suite de l'environnement d'ordonnancement ainsi que de ses différentes contraintes, le tout dans le but d'optimiser une certaine fonction objectif.

La polyvalence des problèmes d'ordonnancement vient du fait que les tâches peuvent représenter des concepts aussi variés que des commandes, des programmes informatiques ou encore des avions en attente d'atterrissage alors que les machines peuvent modéliser des machines industrielles, des processeurs ou encore des terminaux d'aéroport.

Dans notre cas, nous nous intéressons à l'ordonnancement d'ateliers. Chaque tâche $j$, $1 \leq j \leq n$ est composée de $m$ opérations notées comme suit $\{O_{1j}, O_{2j}, \ldots, O_{mj}\}$, une opération $O_{ij}$

correspond à l'exécution de la tâche $j$ par la machine $M_i$, $1 \leq i \leq m$.

Tout d'abord, nous commençons par le problème du flow shop. Dans ce dernier, l'ordre de passage des tâches à travers le système de machines est identique : toutes passent par la machine $M_1$, la machine $M_2$ et ainsi de suite jusqu'à la machine $M_m$. Si de plus, la séquence de tâches sur chaque machine est identique, on parle de flow shop de permutation. Par ailleurs, des temps de réglages peuvent être nécessaires entre l'exécution de deux opérations consécutives sur une même machine. On note $s_{ijk}$ le temps de réglage qui doit s'écouler entre la fin du traitement de l'opération $O_{ij}$ et le début de l'opération $O_{ik}$.

Ensuite, nous passerons à l'étude du problème du job shop. À l'opposé du flow shop, ici chaque tâche aura sa propre *route* à travers le système. Pour finir, nous passerons au problème de l'open shop dans lequel les routes sont "immatérielles" et où chaque tâche peut prendre n'importe quelle route à travers le système.

Les récentes recherches dans le domaine de l'ordonnancement [Hall et al. (1997, 2000); Sierra et al. (2011); Vahedi-Nouri et al. (2013); Zouba (2009)] ont conclu que les modèles *classiques* étaient devenus peu réalistes car ils ne modélisaient pas assez fidèlement les systèmes réels et qu'il était donc nécessaire d'y ajouter de nouveaux paramètres dans le but de réduire l'écart entre la théorie et la pratique. Ce nouveau pan de la théorie de l'ordonnancement est appelé ordonnancement avec contraintes de ressources humaines (ou contraintes d'opérateurs).

La présente thèse se veut une contribution dans le domaine sus-mentionné.

Le problème de flow shop de permutation avec temps de réglages et pour objectif la minimisation du temps de fin global est connu pour être $\mathscr{NP}$-difficile au sens fort, ce qui justifie l'utilisation d'une approche heuristique pour sa résolution. Nous y considérons que les réglages sont effectués par des opérateurs dont le nombre est égal à celui des machines. Pour

sa résolution, nous avons adapté la métaheuristique Migrating Birds Optimization [Duman et al. (2012)] qui avait fait ses preuves sur le problème du flow shop de permutation. Nous avons présenté une version de base que nous avons ensuite améliorée et déclinée en quatre versions, chacune ayant ses spécificités. Ces quatre méthodes ont permis d'obtenir des solutions structurellement différentes mais d'équivalente qualité ce qui offre de la flexibilité lors de la prise de décision.

Par la suite, nous nous sommes intéressés aux problèmes d'ordonnancement d'ateliers avec un nombre d'opérateurs inférieur au nombre de machines, une analyse de la littérature nous ayant permis de constater que cette configuration était fréquente en pratique. Dans les modèles construits, une opération a besoin d'un opérateur pendant toute la durée de son exécution. Ainsi, à la différence des modèles étudiés dans la littérature, l'impact des opérateurs sur le système est modélisé par les retards engendrés par le nombre réduit d'opérateurs.

Nous étudions deux cas : premièrement, le cas où l'ordonnancement des tâches et l'affectation d'opérateurs se fait simultanément, deuxièmement, le cas où un ordre des tâches est donné au départ. Nous étudions également deux modes de changement d'affectation des opérateurs. Premièrement, le mode de changement libre, où un opérateur peut interrompre l'opération de laquelle il est en charge à tout moment pour aller traiter une autre ou simplement rester inactif. L'opération interrompue pourra par la suite être complétée par ce même opérateur ou un autre. Deuxièmement, le mode de changement en fin de tâche, où un opérateur ne peut interrompre une opération dont il est en charge avant la fin de celle-ci. Il pourra ensuite rester sur la même machine ou changer de machine.

Pour chacun des problèmes sus-mentionnés, nous traitons deux objectifs. Le premier, noté $C_{max}$, représente le temps de fin global de l'ordonnancement ou makespan, autrement dit, le temps auquel la dernière opération aura quitté le système. Pour le second, nous supposons

que les tâches ont chacune une date d'échéance $d_j$ et nous voulons minimiser ce que l'on note $L_{max}$, et qui représente le retard algébrique maximum. Par ailleurs, dans le but d'obtenir des modèles plus réalistes, pour les cas avec mode de changement d'affectation libre, nous considérons des opérateurs ayant des niveaux de performance différents.

Dans notre étude, nous avons adopté la méthodologie classique utilisée en optimisation combinatoire. Tout d'abord, nous avons étudié la complexité de nos problèmes. Pour les cas résolubles en temps polynomial, nous avons exhibé des méthodes de résolution [Benkalai et al. (2016b,a, 2017a, 2018b)]. Pour la résolution de certains cas $\mathscr{NP}$-difficiles, nous avons conçu des heuristiques dont nous avons évalué la qualité à l'aide de bornes proposées [Benkalai et al. (2015, 2017b)]. L'évaluation de la qualité des heuristiques s'est faite après une étude expérimentale. Étant donné qu'il n'existait pas de benchmarks pour notre problème, nous avons adapté celui de Taillard pour le flow shop de permutation en ajoutant les contraintes d'opérateurs.

Le reste de la thèse est organisé comme suit. Le Chapitre 1 présentera certains concepts de base nécessaires à la bonne compréhension de la thèse. Il portera sur la théorie de la complexité, les principales approches de résolution en optimisation combinatoire ainsi que sur la méthodologie de résolution utilisée. Le Chapitre 2 présentera la terminologie de base de la théorie de l'ordonnancement. Une revue de la littérature fera l'objet du Chapitre 3. Le Chapitre 4 portera sur un problème de flow shop avec temps de réglages, le nombre d'opérateur y est égal au nombre de machines. Les Chapitres 5, 6 et 7 seront respectivement dédiés aux problèmes de flow shop, de job shop et d'open shop avec opérateurs où le nombre d'opérateurs est inférieur au nombre de machines. Enfin, nous clôturerons cette thèse avec une conclusion et quelques perspectives pour de futurs travaux.

# CHAPITRE 1

# CONCEPTS DE BASE

## 1.1 NOTIONS DE COMPLEXITÉ

En général, les problèmes d'ordonnancement appartiennent à une classe plus large de problèmes de *recherche* combinatoire. Un problème de *recherche* combinatoire $\Pi$ est un ensemble de paires $(I,A)$ où :

1. $I$ est une instance du problème considéré, en d'autres termes, c'est une affectation de valeurs particulières aux paramètres définissant le problème.
2. $A$ est une réponse (solution) à l'instance $I$.

Nous distinguons deux sous-classes de problèmes de recherche combinatoire, à savoir :

— Les problèmes de décision, dont la solution prendra la forme d'un oui ou d'un non.

— Les problèmes d'optimisation, dont la réponse spécifie une solution qui optimise un certain objectif.

N.B. Dans le reste du document, nous supposons, sauf mention contraire, que nous traitons des problèmes de minimisation.

Pour tout problème d'optimisation combinatoire, il existe un problème de décision associé.

Le codage des problèmes de recherche (de leurs paramètres) est l'un des plus importants aspects abordés lors de la conception de méthodes pour leur résolution.

Généralement, pour coder une instance $I$ d'un problème $\Pi$, nous utilisons une chaîne $x(I)$ de symboles issus d'un certain alphabet [1] $\Sigma$ et combinés suivant les règles d'un schéma de codage particulier $e$. La *taille en entrée* $|I|$ de l'instance $I$ sera alors la longueur de la chaîne $x(I)$. Usuellement, il est intéressant d'exprimer la *taille en entrée* d'une instance $I$ en fonction du nombre d'éléments d'un certain ensemble dont la cardinalité est dominante pour l'instance.

Un algorithme est une procédure servant à résoudre un problème, *i.e.* il s'agira de trouver une réponse (oui ou non) pour les problèmes de décision et une solution optimale pour les problèmes d'optimisation. Dans le cadre de la résolution d'un problème d'optimisation, on peut considérer des méthodes approchées qui fournissent des solutions réalisables qui tendent vers mais ne garantissent pas l'optimalité.

La complexité temporelle d'un algorithme $A$ permettant de résoudre un problème $\Pi$ est une fonction associant à chaque *taille en entrée* d'une instance $I$ de $\Pi$ un nombre maximum d'étapes élémentaires (ou unités de temps) d'un ordinateur nécessaires à la résolution d'une instance de cette taille au moyen de l'algorithme $A$.

Du point de vue de la complexité, il existe deux différentes classes d'algorithmes :

1. Les algorithmes polynomiaux, dont la complexité temporelle est en $O(p(k))$ pour un certain polynôme $p$, $k$ étant la taille en entrée d'une instance. Ces algorithmes sont considérés comme "bons" car ils induisent souvent un temps de calcul assez raisonnable.
2. Les algorithmes exponentiels, dont la complexité temporelle ne peut être bornée

1. Un ensemble fini de symboles.

polynomialement. Leur temps d'exécution peut très rapidement devenir prohibitif avec l'augmentation de la taille en entrée de l'instance considérée. Seulement, il n'est pas toujours possible de résoudre un problème à l'aide d'un algorithme de complexité polynomiale, souvent, seuls des algorithmes exponentiels existent.

En pratique, il n'est pas nécessaire d'avoir la forme explicite de la complexité temporelle d'un algorithme, mais plutôt une borne supérieure qui permettrait de donner une indication sur le comportement de la fonction complexité avec l'augmentation de la taille de l'instance du problème considéré.

Comme spécifié précédemment, nous pouvons associer à tout problème d'optimisation un problème de décision, les deux classes de problèmes pourront être analysées de la même manière car un problème de décision n'est pas plus difficile du point de vue calcul que le problème d'optimisation auquel il est associé. Ceci induit qu'une résolution polynomiale d'un problème d'optimisation induit la résolution polynomiale du problème de décision associé. Inversement, si le problème de décision est "difficile", le problème d'optimisation auquel il est associé le sera aussi.

Les principales classes de problèmes sont comme suit :

1. La classe $P$ qui contient les problèmes de décision ayant un algorithme polynomial pour leur résolution.
2. La classe $NP$ qui contient les problèmes de décision pour qui une réponse positive (oui) peut être vérifiée en temps polynomial. De ce fait, la relation $P \subset NP$ est établie.

La conjecture largement admise de $P \neq NP$ permet de tracer le schéma suivant [Paschos (2005)] :

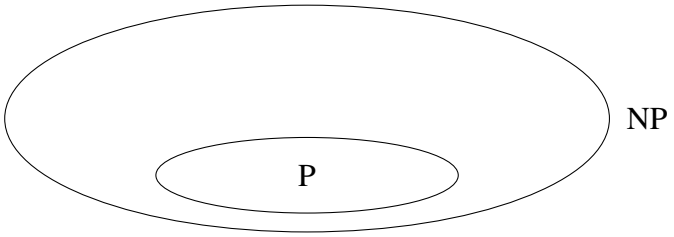



**Figure 1.1: $P$ et $NP$, sous l'hypothèse $P \neq NP$.**

Une transformation polynomiale d'un problème $\Pi_2$ en un problème $\Pi_1$, notée $\Pi_2 \propto \Pi_1$, est une fonction associant à chaque instance de $\Pi_2$ une instance de $\Pi_1$ telle que :

— La réponse à une instance $I_2$ est oui $\forall\ I_2$ de $\Pi_2 \Leftrightarrow$ La réponse à une instance $f(I_2)$ de $\Pi_1$ est oui.

— $f$ est calculable en temps polynomial en la taille en entrée de $I_2$ ($|I_2|$).

Une importante sous-classe de $NP$ est celle des problèmes $NP$-complets. Ces derniers sont des problèmes de $NP$ auxquels se réduisent tous les autres problèmes de $NP$ ($\forall\ \Pi_2 \in NP,\ \Pi_2 \propto \Pi_1$) [Cook (1971)]. La $NP$-complétude d'un problème est prouvée en réduisant un autre problème, reconnu $NP$-complet, à ce dernier. Un problème d'optimisation est dit $NP$-dur si son problème de décision associé est $NP$-complet. D'un autre côté, pour prouver qu'un problème est "facile" (appartenant à $P$), il suffit de construire un algorithme polynomial le résolvant. Ceci peut être fait en le réduisant à un autre problème résoluble en temps polynomial.

Toujours sous l'hypothèse $P \neq NP$, nous avons le schéma suivant [Paschos (2005)] :

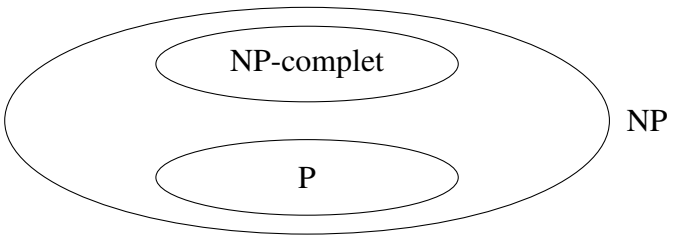



**Figure 1.2: $P$, $NP$ et $NP$-complet, sous l'hypothèse $P \neq NP$.**

Malgré le fait que les problèmes *NP*-complets soient reconnus difficiles, certains peuvent être résolus efficacement en pratique ; ceci est dû au fait que les algorithmes qui les résolvent sont polynomiaux en la taille en entrée $|I|$ ainsi qu'en le nombre maximal $max(I)$ apparaissant dans une instance $I$. Ces algorithmes sont dits *pseudo-polynomiaux*. Ce type d'algorithmes est généralement construit pour des problèmes de *nombres*. Un problème $\Pi$ est dit problème de nombres s'il n'existe pas de polynôme $p$ tel que $max(I) \leq p(|I|)$ pour toute instance $I$ de $\Pi$.

$\Pi_p$ est un sous-problème de $\Pi$ restreint aux instances pour lesquelles $max(I) \leq p(|I|)$ pour un certain polynôme $p$. Ainsi, $\Pi_p$ n'est pas un problème de nombres.

Un problème $\Pi$ sera dit *NP*-complet au sens fort si :

— $\Pi \in NP$.
— Il existe un polynôme $p$ définit sur des entiers pour lequel $\Pi_p$ est *NP*-complet.

Il découle de cette définition que si $\Pi$ est *NP*-complet et n'est pas un problème de nombres, alors $\Pi$ sera dit *NP*-complet au sens fort.

Pour prouver la *NP*-complétude au sens fort d'un problème, on devrait trouver un polynôme $p$ pour lequel $\Pi_p$ est *NP*-complet, ce qui n'est pas très facile, plusieurs travaux suggèrent plutôt l'utilisation des transformations pseudo-polynomiales [Blazewicz et al. (2001)].

## 1.2 APPROCHES DE RÉSOLUTION

Des décennies des travaux intensifs de recherche tous plus inventifs les uns que les autres n'ont toujours pas permis de trouver un algorithme "efficace" pour la résolution des problèmes *NP*-durs d'ordonnancement, il n'en demeure pas moins vrai que beaucoup de progrès et d'avancées ont été accomplis depuis que ces derniers se sont invités dans la cour des problèmes

d'optimisation combinatoire, il y a de cela plusieurs décennies. La suite de cette section offre une vue d'ensemble sur les principales classes de méthodes utilisées en ordonnancement.

### *1.2.1 ALGORITHMES CONSTRUCTIFS*

Ces algorithmes permettent de construire une solution optimale à partir des données du problème en suivant des règles simples qui déterminent l'ordre de traitement des tâches (ces algorithmes sont polynomiaux). Généralement, les méthodes constructives incluent une connaissance du problème qui implique une bonne compréhension de la structure de la solution.

### *1.2.2 MÉTHODES ÉNUMÉRATIVES*

La section suivante sera consacrée aux méthodes énumératives. Les méthodes abordées, à savoir la programmation dynamique et la méthode branch-and-bound, sont toutes deux des méthodes d'énumération implicite, c'est à dire qu'elle considèrent certaines solutions indirectement, sans les évaluer explicitement.

**La programmation dynamique**

La programmation dynamique est une méthode d'énumération qui trouve ses origines dans les travaux de Bellman de 1957 [Bellman (1957)]. Cette méthode est applicable à tout problème pouvant être décomposé en une séquence de sous-problèmes imbriqués de telle manière que la solution de l'un de ces sous-problèmes dépend de la solution de celui qui le précède. La méthode repose sur le principe d'optimalité de Bellman [2], ce qui permet de décrire le critère

2. qui stipule que toute partie d'une solution optimale est optimale pour le problème restreint qu'elle défini.

d'optimisation sous forme d'équation récursive.

**Branch-and-bound**

Soit $M$ une borne inférieure de la fonction objectif du problème considéré ; le fait de trouver une solution réalisable d'évaluation $M$ signifie que cette dernière est optimale. Ainsi, le fait de borner la valeur de la fonction objectif permet de réduire -parfois substantiellement- l'espace de recherche. Ceci combiné à l'adage : "diviser pour mieux régner" donne le principe du Branch-and-Bound (Séparation et Évaluation).

L'idée du Branch-and-Bound, méthode d'énumération implicite, consiste à diviser le problème principal -dont les solutions forment un ensemble $S$- en sous-problèmes, généralement deux à deux disjoints, qui seront éventuellement eux-mêmes divisés par la suite de la même manière après leurs résolutions engendrant ainsi des sous-ensembles de plus en plus petits de $S$. Ces derniers sont considérés comme les ensembles de solutions des sous-problèmes du problème original. L'étape d'évaluation (bounding) calcule une borne inférieure pour chaque sous-problème généré.

L'ensemble de sous-problèmes d'un processus branch-and-bound est généralement représenté sous forme d'une arborescence qui contient :

— Au premier niveau un nœud unique représentant le problème initial.
— Aux niveaux suivants des nœuds représentant les sous-problèmes générés au fil de l'exécution de la méthode.

Différentes étapes sont nécessaires à la construction de l'arborescence de recherche, à savoir :

— La nécessité de pouvoir partitionner l'ensemble des solutions réalisables en des ensembles de plus en plus petits, jusqu'à obtenir des ensembles contenant un seul

ordonnancement.

— Une procédure de calcul de borne inférieure pour tout nœud de l'arborescence.

— Une solution dite "*de test*" qui est construite au bout d'un certain point de l'avancement de l'algorithme et dont l'évaluation sert de borne supérieure.

— La stratégie de parcours de l'arborescence.

Les nœuds *actifs* de l'arborescence correspondent aux sous-problèmes non encore éliminés et dont les propres sous-problèmes n'ont pas encore été générés. Un nœud sera stérilisé (éliminé) si la valeur de sa borne inférieure est supérieure à $\gamma(s)$, $\gamma(s)$ étant l'évaluation d'une solution s trouvée au cours de l'exécution de la méthode [3].

Dans le processus de conception d'un algorithme de type branch-and-bound, les points suivants devraient être considérés :

— La procédure de branchement et la stratégie d'exploration de l'arborescence.

— La procédure de calcul des bornes inférieures et le critère de stérilisation des nœuds.

### *1.2.3 PROGRAMMATION MATHÉMATIQUE*

La programmation mathématique comprend une famille de techniques servant à optimiser une fonction sous certaines contraintes sur des variables indépendantes.

Après modélisation, un modèle mathématique peut alors être résolu en utilisant les algorithmes développés à cet effet ; un ordonnancement optimal sera déduit de la solution optimale obtenue.

En général, des méthodes de branch-and-bound ou alors des méthodes de coupes [4] sont

3. elle sert de borne supérieure aux différents sous-problèmes.

4. Proposées à l'origine par Dantzig *et al.* pour le problème du voyageur de commerce [Dantzig et al. (1954)], elle suggèrent la résolution, au départ, d'une relaxation d'un certain problème, puis l'ajout, d'itérations en itérations, d'inégalités spécifiques visant à réduire l'espace de solutions sans éliminer de solutions du problème initial : des coupes.

utilisées pour la résolution de programmes linéaires en nombres entiers.

Les méthodes présentées dans les deux dernières sections (1.2.2, 1.2.3) se basent sur une énumération (souvent implicite) de l'ensemble des solutions réalisables du problème à résoudre. Cette dernière peut être améliorée par des techniques permettant de détecter le plus tôt possible les zones de recherche stériles -à éliminer, et ce par l'établissement de bornes inférieures et supérieures pouvant être fournies respectivement par des relaxations et des méthodes approchées. Ceci dans l'espoir de mieux cerner de bonnes zones de recherche, et ainsi guider la procédure d'énumération des solutions et accélérer la convergence de l'algorithme.

Ce qui nous empêche de considérer toutes les solutions une à une est leur nombre ; il est clair qu'à partir d'une certaine taille, le nombre de solutions possibles pour un problème d'ordonnancement ainsi que le temps nécessaire à leur évaluation deviennent excessivement grands ; il s'agit du phénomène de l'explosion combinatoire. En conclusion, un algorithme de résolution efficace devrait la permettre sans explorer toutes les solutions possibles, seulement, la conjecture largement admise $P \neq NP$ suggère que pour beaucoup de problèmes difficiles d'ordonnancement, un tel algorithme ne pourra probablement jamais être trouvé.

Cependant, il faut savoir que dans de nombreux contextes d'applications pratiques avec un intérêt stratégique, de bonnes solutions sont nécessaires dans un court laps de temps (obsolescence, concurrence, etc.). Le recours aux méthodes dites approchées se présente comme une option nécessaire, en réponse à un certain nombre de contraintes liées à l'évolution des paramètres des-dits problèmes, à l'urgence d'obtention de solutions réalisables, etc.

### *1.2.4 MÉTHODES APPROCHÉES*

Les méthodes approchées sont des algorithmes permettant de trouver une ou plusieurs solutions réalisables pour un problème donné. Elles sont également appelées méthodes incomplètes, et ce, car elles "troquent" la complétude [5] contre l'efficacité [6]. En effet, ces méthodes sont caractérisées par leur "rapidité", ce qui semble remédier à notre problème qui est la "lenteur" des méthodes exactes. Elles sont ainsi très adéquates pour les instances de grandes tailles [7], ainsi que pour celles caractérisées par des données approchées ou dynamiques.

Notons que, ces méthodes possédant pour la plupart une composante aléatoire, leur rapidité est contrebalancée par le fait qu'elle n'offrent strictement aucune garantie d'optimalité pour les solutions qu'elles donnent. Néanmoins, nous verrons que cela ne présente pas de sérieux problèmes. Nombre de méthodes approchées appliquées en ordonnancement donnent des résultats très satisfaisants, parfois même inespérés.

Deux types de méthodes approchées ont été appliquées en ordonnancement : les heuristiques et les métaheuristiques. Les premières sont spécifiques (dédiées) à certaines classes de problèmes, tandis que les secondes sont une sorte d'ossature générale à adapter au problème à résoudre.

Une condition nécessaire à l'application de ce type de méthodes est que leur complexité temporelle soit dans le pire des cas bornée supérieurement par un polynôme d'ordre réduit en la taille en entrée de l'instance.

5. La garantie de l'optimalité.
6. En d'autres termes, elles permettent d'obtenir de bonne solution dans un temps considérablement réduit.
7. Les instances sans propriétés particulières qui pourraient permettre leur résolution efficace.

## Heuristiques

Les heuristiques, du grec *heuriskein* qui signifie trouver, sont des méthodes approchées spécifiques à un problème donné. Elles permettent d'aboutir "rapidement" à une solution réalisable de l'instance du problème à résoudre.

Même si elles sont relativement efficaces en pratique, ces méthodes ont pourtant leurs limites. En plus d'être spécifiquement dédiées à un problème donné, les heuristiques sont particulièrement caractérisées par le fait de converger, voire d'être piégées dans des minimums locaux. Pour remédier à cela, de nouvelles approches de résolutions, dont la conception est totalement différente ont vu le jour : Les Métaheuristiques.

## Métaheuristiques

Longtemps désignées comme étant des "heuristiques modernes", les métaheuristiques, du grec *heuriskein* qui signifie "trouver" et *meta* qui signifie "de niveau supérieur", furent nommées ainsi par Fred Glover en 1986 [Blum et Roli (2008)]. Elles sont, en quelque sorte, des méthodes heuristiques pour la génération de méthodes heuristiques [Cook (2012)] ; il s'agit d'une sorte d'ossature générale (framework) caractérisée par un ensemble de concepts fondamentaux qui lui sont propres ; elles possèdent toutes un haut niveau d'abstraction, ce qui leur permet d'être utilisées pour la résolution d'un large éventail de problèmes, et ce, en procédant par leur adaptation à l'instance du problème à résoudre. Elles requièrent des techniques permettant de trouver rapidement des solutions de bonne qualité dans des espaces de recherche très grands et très complexes. Ces méthodes utilisent souvent des caractéristiques spécifiques du problème dans le but d'améliorer et d'accélérer la convergence, si ces caractéristiques ne sont pas disponibles au départ, un processus d'apprentissage permettra d'accumuler dynamiquement

de l'information pendant le processus de recherche [Blazewicz et al. (2001)].

L'existence des métaheuristiques de manière indépendante des problèmes qu'elles résolvent a permis d'élargir les horizons de recherche et de puiser de nouvelles idées dans des domaines très différents ; ainsi, des chercheurs appartenant à des domaines scientifiques très variés ont pu apporter leur contribution.

Au-delà de cela, les métaheuristiques ont en commun le fait que toutes permettent une plus large exploration de l'espace des solutions car chacune possède un mécanisme propre qui lui permet, par le biais d'une utilisation intelligente et biaisée de l'aléatoire (se basant sur la mémoire, l'expérience, ou la descente), d'échapper aux optimums locaux. Les métaheuristiques sont principalement caractérisées par leur simplicité d'implémentation, ce qui leur a valu de recevoir une attention considérable, sans cesse croissante, depuis environ une quarantaine d'années.

Deux principaux ingrédients "opposés" à prendre en considération lors de l'élaboration d'une métaheuristique sont l'*intensification*, qui est une recherche concentrée sur les zones considérées prometteuses en vue de les exploiter au mieux et la *diversification* qui quant à elle permet une plus large exploration des différentes parties de l'espace de recherche.

Différents éléments sont nécessaires à la mise en œuvre d'une métaheuristique, à savoir, le choix de la solution de départ, le choix du voisinage, la manière d'explorer ce voisinage, etc.

On en distingue deux principales classes : les métaheuristiques à trajectoire (monosolution ou à solution unique qui sont plutôt intensificatrices) et les métaheuristiques évolutives (ou à population de solutions qui, elles, sont plutôt diversificatrices). Pour plus de détails et une vue d'ensemble des principales métaheuristiques, voir [Talbi (2009); Fister et al. (2013)].

**Algorithmes d'approximation**

Les algorithmes d'approximation sont des heuristiques dont la précision est évaluée analytiquement. Plusieurs ratios d'évaluation de performance existent dans la littérature, nous citons pour l'exemple :

— Le ratio d'un algorithme d'approximation $A$ : $R_A(I) = \frac{A(I)}{OPT(I)}$, où $A(I)$ est la valeur de la fonction objectif de la solution fournie par $A$ pour l'instance $I$ et $OPT(I)$ est l'évaluation de la solution optimale de l'instance $I$.

— Le ratio de performance absolu :

$$R_A = inf\{r \geq 1 | R_A(I) \leq r \text{ pour toutes les instances de } \Pi\}$$

L'analyse au pire cas d'un algorithme d'approximation est faite en général en évaluant la convergence des deux erreurs suivantes vers 0, voir [Blazewicz et al. (2001)] pour plus de détails sur les différents types de convergence.

— L'erreur absolue : $a_n = A(I_n) - OPT(I_n)$.

— L'erreur relative : $b_n = \frac{a_n}{OPT(I_n)}$

On se demanderait alors s'il existe un algorithme offrant une meilleure garantie de performance ? On chercherait un tel algorithme en cherchant à compenser les limites du précédent, et ainsi de suite chercher à améliorer à chaque fois la garantie de performance.

Une autre manière de faire, tellement plus méthodique est très présente dans la littérature, il s'agit des schémas d'approximation. Ces schémas permettent de trouver, étant donné une précision $\varepsilon$, un algorithme $A$ avec une garantie de performance $R_A(I)$, telle que :

$$R_A(I) \leq 1+\varepsilon, \ \ \forall\, I \text{ instance d'un problème } \Pi.$$

Nous citons pour l'exemple :

— Les schémas *PTAS* (Polynomial Time Approximation Scheme) qui sont des schémas d'approximation générant des heuristiques polynomiales en la taille en entrée de l'instance considérée.

— Les schémas *FPTAS* (Fully Polynomial Time Approximation Scheme) qui sont des schémas d'approximation générant des heuristiques polynomiales en la taille en entrée de l'instance considérée et en la précision désirée $\frac{1}{\varepsilon}$.

## 1.3 MÉTHODOLOGIE DE RÉSOLUTION D'UN PROBLÈME D'OPTIMISATION COMBINATOIRE DÉTERMINISTE

Les problèmes d'ordonnancement déterministes faisant partie des problèmes d'optimisation combinatoire déterministes, l'approche générale d'analyse de ces problèmes peut leur être appliquée, tout en prenant en compte cependant leurs particularités.

Souvent, le temps alloué à la résolution d'un problème d'ordonnancement est très limité, ainsi, seuls les algorithmes polynomiaux d'ordre réduit sont d'un intérêt pratique. L'examen de la complexité des problèmes considérés est ainsi à la base de leur analyse.

Si un problème appartient à *P*, un algorithme polynomial doit avoir été trouvé, auquel cas une recherche d'algorithmes pourra être orientée vers l'amélioration de la complexité existante.

Dans la démarche d'analyse d'un nouveau problème d'optimisation, s'il n'est pas dans *P*, il faut d'abord voir s'il a déjà été prouvé qu'il était $\mathscr{NP}$-dur, et dans le cas contraire, tester son

appartenance à cette classe de problèmes. Si nous prouvons que le problème considéré est $\mathscr{NP}$-dur, il faudra alors admettre que pour les plus grandes instances, il ne sera pas possible de trouver une solution optimale sans passer un temps excessivement grand pour l'obtenir. Les transformations polynomiales construites entre les différents problèmes existants sont souvent très utiles lors de l'analyse de nouveaux problèmes.

Cependant, en pratique, un problème ne peut être laissé sans solution. L'on utilisera alors des heuristiques ou algorithmes d'approximation qui produisent des solutions réalisables, souvent de très bonne qualité, mais sans en garantir l'optimalité. Bien entendu, les heuristiques ne seront utilisées que si la taille de l'instance du problème considéré rend l'utilisation de toute méthode exacte infaisable. En fait, plusieurs possibilités s'offrent à nous :

— Relaxer certaines contraintes du problème et résoudre le problème relaxé (par exemple considérer des temps de traitement unitaires ou encore une certaine forme de contraintes de précédence). La solution obtenue peut être une bonne approximation de la solution du problème original.
— Utiliser des algorithmes d'approximation polynomiaux d'ordre réduit dont l'évaluation est effectuée au travers d'analyses au pire cas, de taux de convergence ou encore d'analyse du comportement moyen.
— Utiliser des méthodes exactes. Dans le cas d'un problème *NP*-dur au sens faible, un algorithme pseudo-polynomial peut être construit. Ce dernier pourra fournir de bons résultats en pratique pour les problèmes de tailles moyennes, les "purs" algorithmes exponentiels étant exclus vue la complexité du problème.

La méthodologie de résolution est résumée par la Figure 1.3 [Blazewicz et al. (2001)].

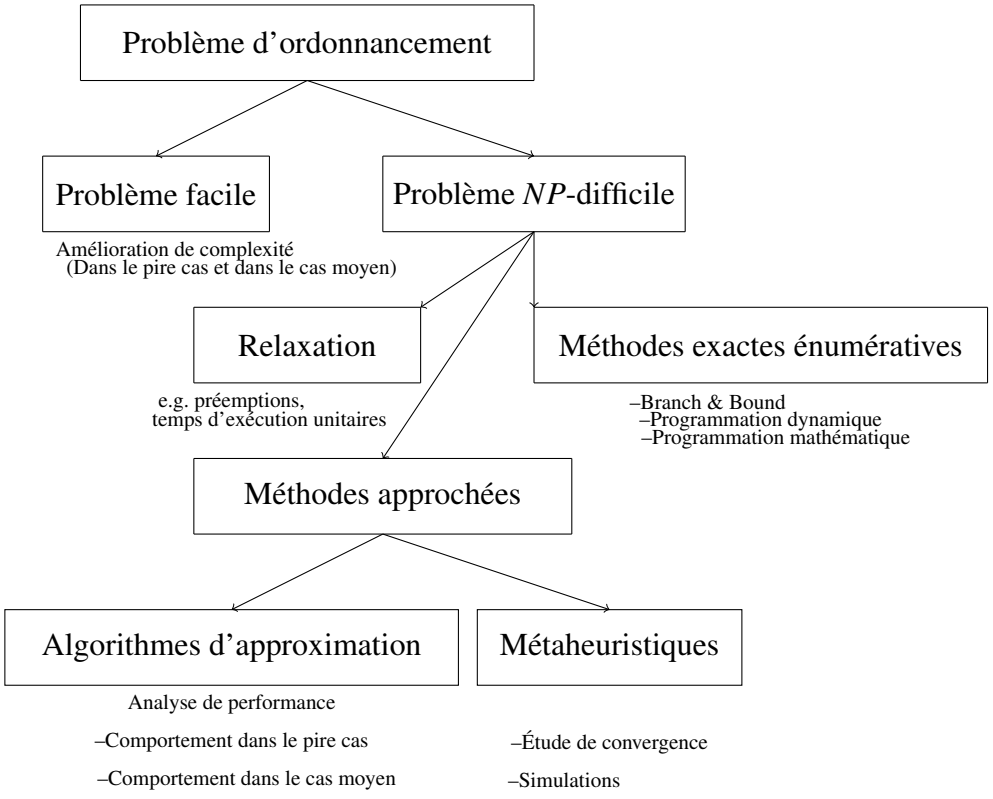



**Figure 1.3: Analyse d'un problème d'ordonnancement.**

# CHAPITRE 2

# GÉNÉRALITÉS SUR LA THÉORIE DE L'ORDONNANCEMENT

## 2.1 INTRODUCTION

Un problème d'ordonnancement est un processus de décision qui vise à allouer certaines ressources (souvent disponibles de manière insuffisante) à des tâches, et ce dans le but d'optimiser un ou plusieurs objectifs. La littérature foisonnante dédiée à ces problèmes atteste de leur grande importance qui est en partie due au grand nombre de leurs applications pratiques, notamment dans le domaine de l'industrie. En effet, de nombreuses situations rencontrées en pratique -telles que la gestion d'un chantier de construction, l'exécution de programmes informatiques sur un ordinateur, l'organisation des départs et arrivées dans un aéroport ou encore la réalisation de produits industriels- peuvent être modélisées en tant que problèmes d'ordonnancement.

Il existe deux types d'ordonnancement, l'ordonnancement déterministe, où un nombre fini de tâches doivent être ordonnancées avec un ou plusieurs objectifs à optimiser, et l'ordonnancement stochastique, où nous avons toujours un nombre fini de tâches à ordonnancer,

mais ici, à la différence du type précédent, les données concernant les tâches ($p_{ij}$ [1], $r_j$ [2], $d_j$ [3]) peuvent ne pas être connues à l'avance. On ne connait alors que les distributions des données et leurs valeurs réelles ne sont connues qu'après la fin du traitement ; dans ce dernier type d'ordonnancement, l'espérance d'un ou de plusieurs objectif(s) doit être minimisée.

L'ordonnancement commença à prendre de l'importance au début du $XX^{me}$ siècle avec les travaux de Gantt, la recherche centrée autour de ce problème attira aussi bien les informaticiens et les chercheurs opérationnels que les ingénieurs industriels. Les premières publications dans ce domaine parurent au début des années 1950 dans le *Naval Research Logistics Quarterly*, elles contenaient des résultats de Smith, Johnson et Jackson. Dans les années 1960, beaucoup de travaux furent consacrés à la programmation dynamique et au développement de formulations mathématiques pour les problèmes d'ordonnancement. Les années 1970 furent quant à elles, après la publication majeure de R. Karp sur la théorie de la complexité [Karp (1972)], plutôt consacrées à l'étude de la complexité des différents problèmes d'ordonnancement ainsi que de leur hiérarchie. Les années 1980 virent naître quant à elles un intérêt grandissant pour les problèmes d'ordonnancement stochastiques, des systèmes d'ordonnancement furent également développés pour la génération de plannings en pratique. L'intérêt théorique des problèmes d'ordonnancement conjugué à leurs applications pratiques expliquent la littérature foisonnante qui leur a été dédiée à ce jour.

## 2.2 DESCRIPTION DE MODÈLES ET NOTATIONS

Dans le présent mémoire de thèse, nous allons uniquement traiter des problèmes d'ordonnancement déterministes. Pendant le dernier demi-siècle, un foisonnement de travaux scientifiques

1. le temps de traitement de la tâche $j$ sur la machine $i$. L'indice $i$ est omis si ce temps ne dépend pas de la machine sur laquelle la tâche est traitée.
2. la date de début au plus tôt de la tâche $j$.
3. la date de fin au plus tard de la tâche $j$ (date d'échéance).

fût dédié à ce type de problèmes, ce qui engendra un grand nombre de modèles.

Nous présentons dans ce qui suit la notation utilisée pour la description des principaux modèles d'ordonnancement déterministes.

En ordonnancement déterministe, un ensemble fini de $n$ tâches, $J = \{1, 2, \ldots, n\}$ doit être ordonnancé sur une ensemble de $m$ machines, $M = \{M_1, M_2, \ldots, M_m\}$, avec un ou plusieurs objectifs à optimiser, les données concernant les tâches sont :

— $p_{ij}$ : le temps de traitement de la tâche $j$ sur la machine $i$. L'indice $i$ est omis si ce temps ne dépend pas de la machine sur laquelle la tâche est traitée.
— $r_j$ : la date de début au plus tôt de la tâche $j$.
— $d_j$ : la date de fin au plus tard de la tâche $j$ (date d'échéance).
— $w_j$ : est le facteur de priorité de la tâche $j$

Un problème d'ordonnancement est décrit par un triplet $\alpha|\beta|\gamma$. Le champ $\alpha$ décrit l'environnement de machines et contient une seule entrée. Le champ $\beta$ fournit des détails sur les caractéristiques et contraintes de traitement et peut contenir plusieurs entrées ou bien aucune ; quant au champ $\gamma$, il représente l'objectif à minimiser et ne contient en général qu'une seule entrée.

### *2.2.1 MODÈLES DE BASE*

Il existe plusieurs modèles d'ordonnancement aux caractéristiques très différentes qui permettent de modéliser un large éventail de scénarii pouvant être rencontrés dans des problèmes pratiques.

**Environnement de machines**

Une opération désigne le traitement d'une tâche par une machine donnée. Les possibles entrées du champ $\alpha$ sont les suivantes :

- 1 pour une seule machine. Ces modèles furent très étudiés, ils possèdent certaines propriétés spéciales, et leur étude fournit plusieurs résultats ainsi qu'une base pour le développement d'heuristiques pour des modèles plus complexes.
- $P_m$ pour $m$ machines parallèles identiques.
- $Q_m$ aussi appelé environnement à machines uniformes, possède $m$ machines parallèles avec des vitesses différentes, la vitesse d'une machine $i$ sera notée $v_i$.
- $R_m$ contient $m$ machines parallèles différentes. Une machine $i$ traitera la tâche $j$ à une vitesse $v_{ij}$.
- $F_m$ représente un environnement à $m$ machines en séries appelé *Flow shop*. Toute tâche $j$ doit être traitée par chacune des $m$ machines, et les $n$ tâches doivent toutes suivre la même route (le même ordre de machines lors du traitement). Après fin du traitement sur une machine, la tâche rejoindra une file d'attente (souvent opérant selon le principe FIFO (premier arrivé, premier servi)) pour la machine suivante, le champ $\beta$ contiendra alors l'entrée *prmu* (permutation flow shop).
- $FF_c$ représente l'environnement *Flow shop flexible* qui est une généralisation du précèdent. Ici, nous avons $c$ étapes en série -chacune contenant un ensemble de machines parallèles- par lesquelles doivent passer les $n$ tâches (toujours dans le même ordre). A chaque étape, une tâche $j$ doit être traitée sur une des machines parallèles, peut importe laquelle.
- $J_m$ représente l'environnement *Job shop* à $m$ machines. Chaque tâche a sa route prédéfinie à suivre. Une distinction est faite entre les Job shops où chaque tâche visite

chaque machine au plus une fois ou plus d'une fois (auquel cas, le champ $\beta$ contiendra l'entrée *recrc* (recirculation).

— $FJ_c$ représente l'environnement *Job shop flexible* qui est une généralisation du précèdent. De manière analogue au $FF_c$, à la place de $m$ machines, il y a $c$ postes de travail ayant chacun un nombre de machines parallèles identiques. Chaque tâche a sa propre route à suivre à travers le système, à chaque poste de travail où elle doit être traitée, elle le sera par n'importe quelle machine de ce dernier. Si une tâche traverse un poste de travail plus d'une fois, le champ $\beta$ contiendra l'entrée *recrc* (recirculation).

— $O_m$ dans un Open shop, chaque tâche $j$ doit être traitée à nouveau sur chacune des $m$ machines (certains de ces temps de traitement peuvent être égaux à zéro). Il n'y a pas de restrictions concernant la route de chaque tâche $j$.

**Caractéristiques de l'environnement**

Le champ $\beta$ peut contenir les entrées suivantes :

— $r_j$ qui indique que les tâches ont des dates de début au plus tôt avant lesquelles elles ne peuvent entamer leur traitement. Si cette entrée est absente du champ $\beta$, les tâches peuvent être disponibles à n'importe quel moment.

— $s_{jk}$ qui indique qu'il existe un temps nécessaire à l'installation de la tâche $k$ après la tâche $j$ qui est dépendant de la séquence selon laquelle les tâches sont traitées.

— *prmp* qui indique que le traitement d'une tâche peut être interrompu avant d'être repris soit sur la même machine, ou sur une autre machine (dans le cas des machines parallèles). Le traitement déjà effectué ne sera pas perdu, la tâche restera sur la machine pendant le temps restant de son traitement.

— *prec* qui indique qu'il existe des contraintes de précédence entre les tâches. Ces

contraintes peuvent avoir des structures particulières comme des chaînes par exemple (chaque tâche a au plus un prédécesseur et au plus un successeur).

— *brkdwn* qui précise que les machines ne sont pas disponibles en permanence. Ici, ces périodes d'indisponibilité seront supposées fixes (dues à de la maintenance par exemple).

— *prmu* qui indique dans les flow shops que les files d'attente entre deux machines sont gérées selon le principe $FIFO$.

— *block* qui indique dans les flow shops un blocage, c'est un phénomène dû à un espace intermédiaire limité entre deux machines successives et qui ferait que la machine en amont ne pourrait livrer la tâche terminée tant que la machine en aval n'aura pas fini le traitement de la tâche précédente.

— $M_j$ qui indique, dans un environnement à machines parallèles que seul un certain ensemble de machines peut traiter la tâche $j$.

— *nwt* (*no-wait*) qui indique un évènement pouvant se produire dans les flow shops, et où une tâche $j$ ne peut attendre entre deux machines successives, sa date de début doit donc être différée de façon à ce qu'elle puisse traverser le système sans aucun temps d'attente. Ce type de contraintes peut être rencontré dans la fabrication de pièces métalliques par exemple ; une pièce chauffée ne devrait pas attendre (au risque de refroidir) avant d'être façonnée.

— *recrc* qui indique, dans les job shops ou job shops flexibles, qu'une tâche peut passer par une machine ou un poste de travail plus d'une fois.

— Tout autre entrée du champ $\beta$ est auto-explicative, par exemple, $d_j = d$ indique que toutes les dates de fin au plus tard sont identiques.

**Critères d'optimisation**

Le champ $\gamma$ précise l'objectif à minimiser qui est une fonction des temps de fin de traitement des tâches qui dépendent de l'ordonnancement de ces dernières. $C_{ij}$, qui est le temps de fin de traitement du job $j$ par la machine $i$. $C_j$, qui est le temps où la tâche $j$ quitte le système (celui où elle quitte la dernière machine sur laquelle elle requiert un traitement). Les objectifs peuvent également être une fonction des dates de fin au plus tard $d_j$, comme ceux exprimant les retards : $L_j = C_j - d_j$ (lateness) , $T_j = max(C_j - d_j, 0)$ (tardiness), la différence entre eux est que $T_j$ n'est jamais négatif.

La pénalité unitaire de la tâche $j$ est définie comme suit :

$$U_j = \begin{cases} 1 & \text{si } C_j > d_j \\ 0 & \text{sinon} \end{cases}$$

Plusieurs objectifs peuvent être minimisés dans un problème d'ordonnancement, dans ce qui suit, nous en citons quelques uns :

— $C_{max}$ ou *makespan* qui est défini comme $max(C_1, C_2, .., C_n)$. Il représente le temps de fin de traitement de la dernière tâche à quitter le système. Un makespan minimum implique en général une grande utilisation des machines.

— $L_{max} = max(L_1, L_2, .., L_n)$ qui mesure la pire violation des dates d'échéance. En pratique, il peut être intéressant de minimiser cette fonction, dépendamment de la politique d'une certaine société. Minimiser $L_{max}$ induirait un planning qui aurait potentiellement plusieurs retards mais qui seraient néanmoins de moindre amplitude.

— $\sum w_j C_j$, la somme pondérée des dates de fin de traitement des $n$ tâches, qui donne une indication sur les coûts d'exploitation ou d'inventaires encourus par le planning.

— $\sum w_j T_j$, la somme pondérée des retards, qui est une fonction plus générale que la précédente.

— $\sum w_j U_j$, la somme pondérée des pénalités, qui est une mesure souvent utilisée en pratique car indicatrice de performance.

Les précédentes fonctions sont toutes non-décroissantes en $C_1, C_2, .., C_n$ (régulières).

## 2.3 CLASSIFICATION ET HIÉRARCHIE DES PROBLÈMES

Un foisonnement de travaux et d'efforts considérables de la part de la communauté scientifique ont permis d'établir une hiérarchie des problèmes d'ordonnancement selon leur complexité, les différentes réductions existant entre eux ainsi que des liens entre les différentes fonctions objectif. Une partie de cette dernière est présentée dans les Figures 2.1, 2.2, 2.3 [4] ; pour plus de détails, voir [Pinedo (2002)].

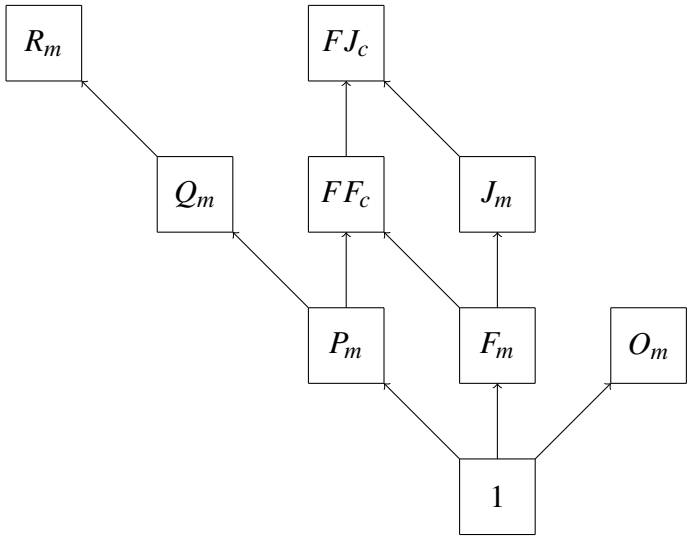
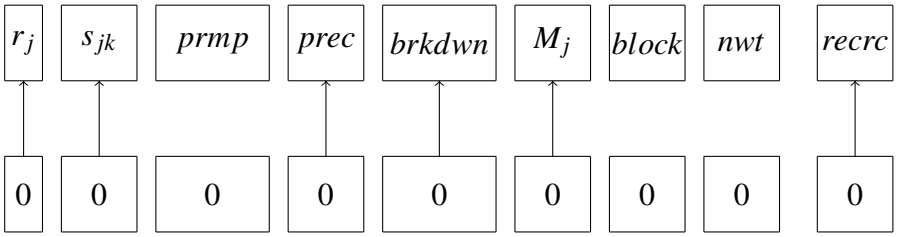



**Figure 2.1: Hiérarchie de complexité de problèmes d'ordonnancement déterministes. Environnements de machines.**

4. Où les flèches indiquent le sens de la réduction.




N.B. Le 0 représente l'absence de contraintes.

**Figure 2.2: Hiérarchie de complexité de problèmes d'ordonnancement déterministes. Caractéristiques de l'environnement.**

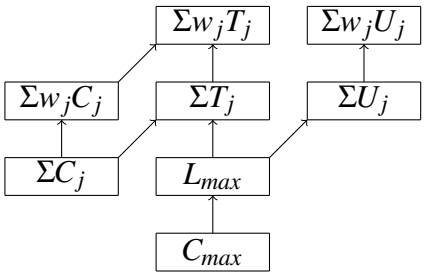



**Figure 2.3: Hiérarchie de complexité de problèmes d'ordonnancement déterministes. Fonctions objectif.**

Il existe également différentes classes d'ordonnancements les principales sont comme suit : les ordonnancements sans retard, où aucune machine ne doit rester inactive si une tâche est disponible pour un traitement, les ordonnancements actifs à partir desquels il n'est pas possible de construire un autre ordonnancement en changeant l'ordre des tâches avec au moins une tâche se terminant plus tôt sans retarder les autres tâches, et enfin, les ordonnancements semi-actifs, où il n'est pas possible qu'une tâche $j$ se termine plus tôt sans changer l'ordre de traitement des tâches sur une des machines.

## 2.4 ORDONNANCEMENT AVEC OPÉRATEURS

Les problèmes d'ordonnancement avec opérateurs s'inscrivent dans la sphère plus large des problèmes d'ordonnancement avec contraintes de ressources. Ces derniers sont en général plus compliqués, parce que chaque tâche peut nécessiter pour son traitement, en plus des machines disponibles dans l'environnement considéré, un certain nombre de ressources supplémentaires [5], qui, dépendamment de leur nature peuvent être classées comme suit :

- La classification par *type* de ressources prend en compte les fonctions assurées par ces ressources.
- La classification par *catégorie* de ressources prend en compte les deux points suivants :
  - Les contraintes sur les ressources. Les ressources peuvent être :
    - *Renouvelables* : auquel cas uniquement leur disponibilité temporaire à tout moment est soumise à des contraintes.
    - *Non-renouvelables* : auquel cas leur disponibilité intégrale jusqu'à un certain point est soumise à des contraintes.
    - *Doublement contraintes* : auquel cas, les deux disponibilités précédemment citées sont soumises à des contraintes.
  - La divisibilité des ressources. Les ressources peuvent être :
    - *Discrètes* : ce qui veut dire qu'elles ne peuvent être allouées à des tâches qu'en quantités discrètes.
    - *Continues* : ce qui veut dire qu'elles peuvent être allouées en quantités arbitraires, a priori inconnues à partir d'intervalles donnés.

Ce type de problème est d'un grand intérêt en pratique. Les ressources peuvent être des travailleurs ou des outils en industrie, ou encore une mémoire externe ou des canaux dans les

5. Souvent disponibles en nombre insuffisant pour satisfaire la demande de toutes les tâches simultanément.

applications informatiques.

Le schéma de notation présenté ci-dessus sera encore enrichi pour capturer les caractéristiques des problèmes avec contraintes de ressources. Le paramètre $\beta_2 \in \{\varnothing,\ res\ \lambda\,\delta\rho\}$ permet de décrire les ressources supplémentaires.

Si $\beta_2 = \varnothing$, il n'y a aucune contrainte de ressource. Par contre, si $\beta_2 = res\ \lambda\,\delta\rho$, le problème a des contraintes spécifiques sur certaines ressources. $\lambda,\ \delta,\ \rho \in \{.,k\}$, représentent respectivement les types de ressources, les limites des ressources et les besoins en ressources. Ces derniers sont arbitraires s'il sont égaux à ., sinon, ils seront égaux à un certain nombre $k$.

# CHAPITRE 3

# REVUE DE LA LITTÉRATURE

Plusieurs modèles d'ordonnancement ont vu le jour à mesure que les recherches avançaient et de nombreuses formulations ont été proposées. Les méthodes de résolution n'ont pas été en reste. En effet, une profusion d'idées et de concepts différents a été appliquée pour la résolution de problèmes d'ordonnancement, allant des méthodes exactes se basant sur la programmation linéaire aux heuristiques dédiées exclusivement à certains modèles jusqu'aux métaheuristiques qui donnent des résultats de qualité de plus en plus surprenante. Il faut dire que ces différents angles "d'attaque" ne sont pas de trop face à des problèmes dont la plupart sont reconnus très difficiles. La présente section sera consacrée à une revue de la littérature dédiée aux problèmes d'ordonnancement d'ateliers classiques ainsi qu'à leurs versions avec opérateurs.

## 3.1 ORDONNANCEMENT D'ATELIERS

Dans le reste de la thèse, nous allons étudier trois types d'ateliers, à savoir : le flow shop, le job shop et l'open shop.

L'étude des environnements flow shops a une grande importance due au fait qu'en pratique, souvent dans des unités de production, un certain nombre d'opérations doit être effectué sur

toutes les tâches, souvent dans le même ordre. C'est notamment le cas dans les chaînes à produit unique.

D'un autre côté, si nous considérons les chaînes de production à plusieurs, nous constatons que les machines servent à produire plusieurs produits différents, mais que lesdits produits ne suivent pas tous le même chemin à travers le système. Cette configuration est modélisée par les environnements open shop.

Précisons aussi que bien souvent en pratique, les routes des différentes tâches sont immatérielles, et c'est à l'ordonnanceur de les spécifier. Cette propriété particulière est capturée par les environnements Open shops. La plupart des problèmes dans un environnement Open shops sont *NP*-durs.

## 3.2 ORDONNANCEMENT D'ATELIERS AVEC OPÉRATEURS

La production est "une transformation des ressources appartenant à un système productif et conduisant à la création de biens ou de services". Les ressources mobilisées peuvent être des hommes, des matières, des informations, de l'équipement ou encore de l'énergie (comme mentionné dans la définition de l'IIE [1]). En gestion de production, il s'agit de trouver la manière la plus efficace d'utiliser lesdites ressources dans le but de produire les biens ou services demandés.

Lors du processus de gestion de la production, différentes décisions doivent être prises. Elles dépendent du niveau de compétence hiérarchique ainsi que celui de leur agrégation. Ainsi, nous distinguerons les décisions stratégiques (à long terme), les décisions tactiques (à moyen terme) et les décisions opérationnelles (à court terme). Ces décisions ont toutes pour objectif

1. Institute of Industrial Engineers

commun de trouver la meilleure solution satisfaisant les contraintes du problème considéré.

L'homme a depuis toujours un rôle décisif en économie, étant d'un côté à la fois consommateur et producteur, et de l'autre le seul à pouvoir faire face à l'imprévu, et ce, malgré les progrès qu'ont connu les équipements de production.

Dans l'étude des diverses décisions possibles, il s'agit d'étudier également les différentes interactions entre les ressources utilisées : matérielles et humaines. Issues respectivement de la production manufacturière et du secteur des services, les méthodes de planification utilisées en gestion de la production et en gestion des ressources humaines ont longtemps été étudiées et améliorées séparément. Seulement, avec l'augmentation de l'importance de la composante humaine dans les systèmes productifs et la multiplication des règles régissant la gestion de cette dernière, le lien entre l'aspect matériel et humain des problèmes d'ordonnancement en a été renforcé et il est devenu apparent qu'il est plus que nécessaire de prendre en considération ces deux aspects lors de la planification de production ; cependant, le développement de méthodes dans ce domaine a longtemps été freiné par la complexité des problèmes engendrés.

Comme nous l'avons précédemment cité, les problèmes d'ordonnancement actuels ont tout intérêt à prendre en considération la composante humaine ; ceci engendre une grande variété de problèmes aux caractéristiques très différentes qui sont tous plus complexes les uns que les autres. Une approche intégrée prenant en compte les deux aspects trouve son intérêt dans les cas où les deux types de ressources sont imbriqués et interdépendants mais aussi dans les cas où ils sont d'importance équivalente ; sinon, les ressources sont suffisamment indépendantes pour être traitées séparément ou alors l'une des ressources est prépondérante par rapport à l'autre, auquel cas elle est considérée en priorité [Baptiste et al. (2005)]. Bien sûr, la mise en œuvre d'une telle approche nécessiterait la disponibilité des informations concernant les deux types de ressources au moment de la décision (ce qui ajoute à la complexité du problème).

Cette analyse de la littérature nous permet de conclure quant à la nécessité de l'existence d'un service de gestion des ressources humaines dans des organisations employant plus de 10 personnes, et ce, pour permettre à l'organisation en question de "tenir le cap qu'elle s'est tracé". Très simple au départ, cette gestion s'est grandement complexifiée avec l'apparition de la protection sociale et de lois régissant le nombre d'heures de travail. Une bonne gestion des ressources humaines permet d'avoir des systèmes productifs efficients, flexibles et réactifs.

De nos jours, dans les horizons à long et moyen termes, les responsables de gestion des ressources humaines font face à "des problèmes mal structurés, interdépendants et s'appuyant sur des concepts difficiles à traduire dans des modèles quantitatifs". Les approches existantes ne permettent pas de faire face à ces difficultés, de plus la gestion des ressources humaines ne résout qu'une partie du problème de gestion d'une organisation et doit souvent faire appel à des hypothèses très réductrices. Ainsi, tout semble indiquer la nécessité de conception de nouvelles approches de résolution qui prennent le problème sous un angle plus général, comme par exemple des approches intégrées.

Par contre, dans les problèmes à court terme, les gestionnaires de ressources humaines font plutôt face à des problèmes répétitifs, bien structurés, mais hautement combinatoires et qui nécessitent des solutions en peu de temps. Là encore, il y a un besoin de nouvelles approches efficaces et rapides permettant de gérer les nombreux aléas qui peuvent se présenter.

En ordonnancement de la production, il s'agit de déterminer l'utilisation la plus efficace d'un ensemble de ressources [2] pour la gestion des flux de produits et de matières en suivant certains procédés de fabrication bien définis, et ce, dans le but de réaliser un ou plusieurs objectifs, comme par exemple la minimisation des coûts. Les ressources en question peuvent être de différents types, à savoir, des machines, des hommes, des outils, etc.

2. toutes soumises à des contraintes de capacité.

Plusieurs décisions doivent être prises lors du processus de gestion des flux, elle sont comme suit :

— **l'éclatement** : découpage de la demande en opérations indépendantes.
— **l'affectation** : affectation des lots aux machines et affectation des hommes aux machines.
— **le séquencement** : détermination de l'ordre de passage des lots sur chacune des ressources.
— **le placement** : détermination des dates exactes de début et de fin de chaque lot.

Un ordonnancement est traditionnellement représenté par un diagramme de Gantt, ayant en abscisse le temps et en ordonnée les ressources. Les opérations y sont représentées par des bandes de longueur proportionnelle à leur durée, comme illustré par la Figure 3.1 ci-dessous.

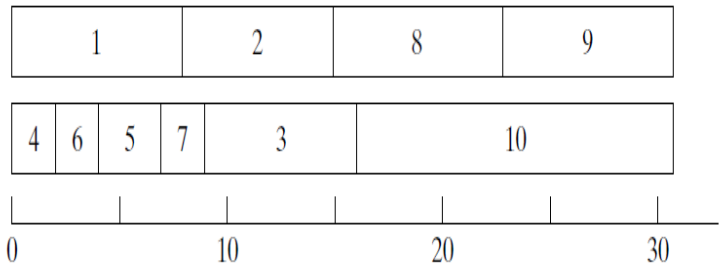



**Figure 3.1: Exemple de diagramme de Gantt pour un problème à 2 machines et 10 tâches.**

Différents types de méthodes de résolution existent pour les problèmes d'ordonnancement. Elle sont comme suit :

1. Par rapport à la nature des données nous avons :
   — **Les approches statiques** : pour des problèmes dont toutes les données sont connues avec certitude.

— **Les approches stochastiques** : pour des problèmes dont les données sont inconnues à l'avance, mais dont on connait les lois de distribution.

2. Par rapport à l'approche de résolution :
   — **Approche distribuée** : où chaque machine est ordonnancée séparément, selon des règles qui lui sont le plus adaptées. Par la suite, l'ensemble sera consolidé en analysant les effets de coordination entre les différentes machines.
   — **Approche globale** : où il s'agit, comme le nom l'indique, d'ordonnancer globalement un ensemble de machines.

Les méthodes globales ont longtemps été destinées aux problèmes de petite taille, mais ceci a beaucoup changé avec l'amélioration de ces méthodes ainsi que des performances des calculateurs ; elles restent cependant appliquées, le plus souvent, à des problèmes bien formalisés.

La gestion des ressources humaines passe par la résolution de problèmes d'horaires et d'affectation de personnel. Dans ce cadre, sont distingués différents type de travaux, à savoir :

— **Les travaux interruptibles** : qui, s'il sont commencé par un opérateur, peuvent être finis par un autre. La considération de travaux interruptibles simplifie grandement le problème.
— **Les travaux non interruptibles** : qui sont chacun réalisés par un même opérateur pendant toute leur durée.

Il existe différents niveaux de décision, à savoir le niveau stratégique, tactique et opérationnel. Il s'agira d'y prendre diverses décisions, à savoir la planification, la rotation de personnel et la gestion des opérations. Ces décisions sont subdivisées par catégories de personnel [3]. Les décisions opérationnelles mettent à jour la planification tactique en fonction des aléas

3. Les catégories de personnel dépendent des fonctions, compétences, etc.

rencontrés.

La confection d'horaires passe par les étapes suivantes :

— La construction de cycles de travail.
— La génération de quarts de travail.
— L'affectation de quarts aux employés.
— L'affectation d'activités aux employés.

Différentes contraintes entrent en jeu lors de l'affectation d'activités aux employés, comme par exemple, les compétences des employés, les réglementation de travail, les types d'activités [4], etc. Les différents travaux présents dans la littérature appuient le caractère capital de l'utilisation des méthodes d'optimisation pour la fabrication d'horaires dans les entreprises. Cependant, vu la complexité de ces problèmes, la recherche est tournée vers l'investigation de nouveaux modèles permettant une meilleure résolution pour des instances de plus grande taille.

### *3.2.1 GESTION DE PROJET*

Un projet est un ensemble de tâches à réaliser au moyen d'un nombre fini de ressource et ce dans le but d'optimiser un ou plusieurs objectifs. Cette définition, assez générale, fait que l'ordonnancement de projet englobe une large palette de modèles différents dont l'ordonnancement de production et la génération d'horaires.

Dans la gestion d'un projet, il faut tout d'abord définir les tâches, les ressources et leurs spécifications (contraintes technologiques et coûts) ; cette définition permettra alors de passer à la définition d'un objectif. Une fois ces définitions établies, les différentes tâches pourront

4. En général, un employé ne devrait pas dépasser une durée maximum pour une activité considérée stressante.

être ordonnancées et le projet réalisé.

Étant donnée la complexité des projets en pratique, ce sont plus souvent des heuristiques ou métaheuristiques qui sont utilisées pour leur gestion. De nombreux travaux sont entrepris ça et là afin de trouver une méthode robuste, i.e. qui donne des solutions de bonne qualité même avec une fluctuation des données du problème.

Le problème de gestion de projet de base admet plusieurs extensions, certains cas considèrent des objectifs autres que la minimisation de la durée, comme la gestion d'un budget par exemple. D'autres cas prennent en compte des contraintes de juste-à temps (pas en avance ni en retard). Les durées des tâches peuvent également varier selon le coût ou l'utilisation de certaines ressources.

La gestion de projet fournit un cadre d'association de gestion de production et de planification des ressources humaines. Face à la complexité de ce problème, les méthodes de l'ordonnancement "classique" sont pour le moins inadaptées et il serait intéressant de concentrer nos efforts dans l'élaboration d'approches permettant de prendre en compte ces deux aspects du problème.

### *3.2.2 APPROCHES INTÉGRÉES À COURT TERME*

Les approches intégrées à court terme prennent en considération à la fois l'aspect matériel mais aussi l'aspect humain dans les systèmes de production. En effet, comme nous l'avons déjà précisé, les ressources humaines occupent une place de plus en plus importante dans les systèmes de production, leurs coûts augmentant proportionnellement à cette importance.

Les ressources humaines jouent un rôle important dans la recherche de flexibilité d'un système productif dans un environnement à paramètres variables. Au niveau opérationnel, les ressources

humaines sont liées aux autres ressources, notamment aux machines, par des contraintes de charge, de capacité mais aussi de disponibilité.

L'intégration des ressources humaines dans les systèmes productifs est un domaine de recherche relativement récent et qui reste largement ouvert. Lors de cette intégration, il s'agira d'attribuer des horaires aux opérateurs mais aussi de les affecter à des postes de travail. Dans ce cadre, l'ordonnancement se base sur les caractéristiques des ressources de différents types utilisées et a pour finalité de déterminer l'état de l'atelier au cours du temps.

Les opérateurs influent sur les systèmes productifs de part leur nombre, leurs compétences, leurs performances et les tâches qu'ils ont à accomplir. Dans un système productif, les durées opératoires sont très souvent susceptibles de changer, elles dépendent des caractéristiques des opérateurs énoncés précédemment, de leurs affectations, mais aussi des machines sur lesquelles les opérations sont traitées ; elles peuvent également évoluer en fonction de l'apprentissage des opérateurs.

En résumé, en considérant les ressources humaines dans le processus de production, nous devons gérer les points suivants :

— La configuration (où sont les opérateurs ?).
— L'affectation (qui fait quoi ?).
— La séquence des différentes tâches sur les machines (dans quel ordre ?).

Avec cette nouvelle vision de l'ordonnancement, il ne faut pas oublier qu'il faut déterminer les règles de passage d'une affectation donnée à une autre.

Lorsqu'un opérateur est partagé entre plusieurs machines, il effectue souvent des opérations répétitives, ceci étant dû au fait que chaque machine a son cycle. Ces cycles sont les plus souvent imbriqués car il est impossible de les réaliser sans retard.

Les auteurs soulignent ensuite que l'affectation d'une machine $M_i$ à un opérateur particulier dépend de la compétence de ce dernier. La compétence d'un opérateur peut être *proportionnelle* (indépendante de la tâche à réaliser), *dépendante* du couple (machine, tâche) ou encore *variable* en fonction du temps (apprentissage, etc.).

Le calcul des durées opératoires dans ce type de problème est particulièrement ardu, il dépend de beaucoup de paramètres eux-mêmes difficiles à évaluer ; en général des moyennes sont utilisées mais ces dernières peuvent s'avérer très différentes des valeurs réelles. La durée devient alors dépendante de la tâche traitée, de la machine sur laquelle elle l'est et de l'opérateur qui y est affecté, mais elle dépend aussi des autres machines du même poste de charge ainsi que des tâches y étant traitées. Ceci renforce encore plus le lien entre la partie "affectation" du problème et sa partie "ordonnancement de tâches". En fait, tout semble plaider en faveur d'approches intégrées traitant simultanément les deux aspects.

Cependant, la description exacte d'une durée opératoire qui engloberait tous ces paramètres est extrêmement complexe et les quelques modèles existants s'appuient sur des hypothèses très simplificatrices afin de pouvoir traiter le problème.

Comme nous l'avons précisé, plus haut, pour connaître notre ordonnancement, il faut connaître les différentes transitions entre les affectations. Les types de transition sont comme suit [Zouba (2009)] :

- **Changement calendaire** : qui s'effectue à la fin de période prédéterminée (journée, quart, etc.)
- **Changement libre** : qui, comme son nom l'indique, peut être effectué à n'importe quel moment.
- **Changement en fin de tâche** : qui permet à un opérateur de changer d'affectation ou de configuration à chaque fois qu'il termine une tâche dont il a la charge.

Par ailleurs, il est important de préciser que l'une des différences fondamentales entre les ateliers avec opérateurs et les cellules robotisées résident dans l'atout de flexibilité qu'apportent les opérateurs.

En conclusion, nous pouvons dire que la prise en compte des ressources humaines est impérative dans les systèmes productifs d'aujourd'hui, cela permet d'améliorer leur flexibilité tout en éliminant certaines dépenses non justifiées. Ces nouveaux problèmes remettent en cause l'ordonnancement "classique" et présentent de nouvelles contraintes (réglementation de travail) et de nouveaux objectifs (coûts). Les modèles existants présentent de nombreuses hypothèses simplificatrices nécessaires pour commencer à aborder ces problèmes de grande complexité. L'ordonnancement avec opérateurs est ainsi un domaine de recherche très ouvert, il a pour objectif, notamment le développement d'approches intégrées reposant sur une modélisation solide de l'acteur humain dans l'environnement de travail.

## 3.3 FLOW SHOP

Auparavant appelés systèmes de production à $k$ niveaux, les flow shop n'ont reçu ce nom qu'à partir des travaux d'Heller en 1960 [Gupta et Stafford (2006)]. Dans le flow shop de base, tel qu'étudié dans l'article de [Johnson (1954)], l'objectif était d'ordonnancer $n$ jobs sur $m$ machines dans le but de minimiser un certain objectif. Les jobs devaient être traités par toutes les machines exactement une fois, et ce en suivant la même route à travers le système. Chaque job était alors traité par la machine $M_1$, la machine $M_2$, et ainsi de suite jusqu'à la machine $M_m$.

Depuis, plusieurs hypothèses furent ajoutées à et/ou retirées du modèle original, ce qui a engendré une multitude de modèles, particulièrement depuis les années 1980 [Gupta et Stafford (2006)].

Le problème du flow shop standard, noté $F_m||C_{max}$, a été démontré $\mathscr{NP}$-difficile au sens fort pour $m \geq 3$ [Garey et Johnson (1979)].

Lorsqu'on le compare au problèmes de flow shop de permutation pour lequel on a compté plus de 1200 publications dans [Gupta et Stafford (2006)], on remarque que le problème du flow shop standard a été relativement peu étudié.

Ceci est principalement dû à la simplicité relative du flow shop de permutation en comparaison avec sa version standard. Il a été prouvé dans [Potts et al. (1991)] que pour certaines familles d'instances, le ratio entre la meilleure solution avec contraintes de permutation lors de la minimisation du makespan et la solution optimale du flow shop standard dépasse $\frac{1}{2}\sqrt{m}$.

Nous rappelons cependant que pour $m \leq 3$, les deux problèmes sont équivalents. Pour plus de détails concernant les méthodes de résolution pour le flow shop standard, voir *e.g.* [Ruiz et al. (2004); Koulamas (1998); Farber et al. (2007); Liao et al. (2007); NoorulHaq et al. (2007); Ying et Lin (2007); Ziaee et Sadjadi (2007); Ying et al. (2010); Vahedi-Nouri et al. (2013)].

Le problème $F_m|prmu, s_{ijk}|C_{max}$, que l'on notera PFS-SDST, est connu pour être $\mathscr{NP}$-difficile au sens fort. L'approche heuristique a été la plus utilisée pour sa résolution. Cependant, seulement quelques métaheuristiques ont été adaptées à cette fin [Ruiz et al. (2004)]. Dans ce qui suit, nous présentons une revue de la littérature des plus importantes études conduites pour la résolution de ce problème. Un état de l'art est disponible dans [Allahverdi et al. (2008)].

Citons tout d'abord l'algorithme génétique et l'algorithme mimétique de [Ruiz et al. (2004)]. Les auteurs comparent l'efficacité de leur algorithme avec celles d'autres approches qui ont fourni de bons résultats pour le problème de flow shop de permutation de base (sans temps de réglages) ainsi que celles d'heuristiques constructives pour le problème PFS-SDST. Ils évaluent l'efficacité de leurs approches sur une version étendue du benchmark de Taillard

pour le flow shop de permutation qui a été fournie par [Vallada et al. (2003)]. Leur algorithme génétique modifie une partie de la population si la recherche stagne [5], et a une sélection par roulette. Les auteurs introduisent également un nouvel opérateur de croisement qui serait plus efficace. Il permet de préserver de bonnes séquences des solutions en copiant des blocs de deux tâches qui apparaissent dans les deux parents directement dans l'enfant et ensuite applique le croisement à deux points de coupures. Il est affirmé dans ce travail que les nouveaux algorithmes surpassent de loin les autres considérés dans leur étude.

[Kumar et Singhal (2013)] proposent aussi un algorithme génétique pour la résolution du même problème avec partition de lots. Cet algorithme utilise une sélection par tournoi et un croisement à deux points de coupure avec une phase de réparation [6]. Son opérateur de mutation repositionne une tâche. Les auteurs testent également les effets de la variation des probabilités de croisement et de mutation sur la qualité des résultats. Leur algorithme parvient souvent à trouver les solutions optimales de leur benchmark plus ou moins rapidement dépendamment des probabilités sus-mentionnées.

L'Iterated Greedy Algorihtm a aussi été adapté pour le problème PFS-SDST par [Ruiz et Stutzle (2005)].Cet algorithme contient deux phases, une phase de destruction qui produit une solution partielle et une phase de construction qui complète cette dernière et remplace la solution actuelle selon un certain critère d'acceptation. Leur étude expérimentale montre que les résultats de cette méthode sont prometteurs.

Pour résoudre le problème de flow shop de permutation avec des familles de temps de réglages, [Lin et al. (2011)] suggèrent l'utilisation d'une approche de recuit simulé à démarrages multiples. L'ensemble de tâches est divisé en familles et les temps de réglages sont nécessaires

5. Si l'algorithme ne parvient pas à améliorer la meilleure solution courante pendant un certain nombre d'itérations donné.

6. Pour réparer les éventuelles solutions non-réalisables.

seulement entre des tâches de différentes familles. Leur approche s'appuie sur les principales propriétés du recuit simulé (*e.g.* convergence efficace et implémentation facile) ainsi que sur celles des stratégies de hill climbing a démarrages multiples (*e.g.* diversification suffisante et échantillonnage efficace de l'espace des solutions voisines). Elle effectue un nombre donné de redémarrages à partir de différentes solutions, offrant ainsi plus de chances de s'échapper des minima locaux. Le voisinage exploré est basé sur des insertions et des échanges de positions. Leur approche a fourni de bien meilleurs résultats que des algorithmes existants dont l'algorithme mimétique de [França et al. (2005)] et la recherche taboue de [Hendizadeh et al. (2008)].

Finalement, nous citons le papier de [Dong et al. (2009)] dans lequel différentes règles de priorité sont présentées dans le but d'améliorer l'efficacité de l'heuristique NEHT-RMB de [Rìos-Mercado et Bard (1998)]. Cette dernière est une adaptation de la fameuse heuristique NEH pour le flow shop de permutation de [Nawaz et al. (1983)]. Leur étude a donné une classification des meilleures heuristiques selon les temps de réglage.

Dans ce qui suit, nous présentons une revue de la littérature dédiée au problème de flow shop avec opérateurs.

Tout d'abord, nous avons remarqué que relativement peu de travaux traitent le problème du cas avec opérateurs. Cela est principalement dû à deux raisons : l'apparition relativement récente du domaine de l'ordonnancement avec ressources humaines et son inhérente difficulté, particulièrement lorsqu'il s'agit de modéliser les différentes caractéristiques des opérateurs telles que leur expérience, leurs aptitudes, la charge de travail variable qui leur est affectée, etc.

Les modèles d'ordonnancement "classiques" supposent que le nombre $k$ d'opérateurs est égal au nombre de machines. Sous cette hypothèse, un opérateur pourra faire fonctionner

exclusivement la machine à laquelle il est affecté. Cependant, cette hypothèse n'est que très rarement vérifiée en pratique. En effet, le nombre d'opérateurs est souvent petit comparé au nombre de machines.

Dans chacune des études dédiées aux problèmes d'ordonnancement avec opérateurs, des modèles basés sur diverses hypothèses ont été proposés.

Depuis les travaux de [Vickson (1980a,b)], la plupart des études ont manipulé des modèles avec des temps de traitement contrôlables par la quantité de ressources assignée aux différents jobs. Lors de la résolution de ces problèmes, on devrait affecter plusieurs opérateurs à un job si on veut en réduire le temps de traitement.

Le modèle proposé dans [Cheurfa (2005)] est différent de ses prédécesseurs. Il décrit un flow shop cyclique avec un nombre d'opérateurs inférieur au nombre de machines. L'auteur y introduit le concept de partage d'opérateurs et étudie un problème avec des opérateurs ayant des aptitudes équivalentes et dont l'intervention se limite à des opérations d'assemblage, de désassemblage et de contrôle.

La littérature dans le domaine de l'ordonnancement avec opérateurs montre qu'une importante partie de ces travaux a été dédiées aux environnements à machines parallèles. On cite pour l'exemple les travaux de [Zouba (2009)]. L'auteur y traite un problème d'ordonnancement des jobs avec affectation simultanée d'opérateurs dont le nombre est inférieur à celui des machines. Ces opérateurs peuvent faire fonctionner simultanément plusieurs machines.

Dans ce modèle, l'auteur étudie l'impact du partage d'opérateurs sur les temps de traitement des jobs. Plusieurs modes d'intervention des opérateurs sont présentés. Dans [Baptiste et Munier (2013)], ce modèle a été amélioré en supposant que le ratio entre les temps de traitement réels et théoriques est une fonction linéaire du taux d'occupation des jobs par les

opérateurs. Même si ce modèle ne permet pas de prévoir les temps de traitement exacts, ce modèle prévoit l'augmentation du makespan qui sera engendrée par le partage d'opérateurs.

L'utilisation du concept "un opérateur-plusieurs machines" était très répandue à la fin des années 1990 et au début des années 2000, particulièrement dans les systèmes de production "juste-à-temps". Ainsi, plusieurs études ont porté sur ledit concept. On cite pour l'exemple le papier de [Cheng et al. (1999)] sur un problème de flow shop à deux machines où l'intervention des opérateurs se limite au réglage des opérations au début et à la fin de celles-ci et où l'objectif est la minimisation du makespan.

Nous citons aussi les travaux de [Baki (1999)]. Ils traitent des problèmes de flow shop et d'open shop avec un seul opérateur. Ce dernier aura à s'occuper d'opérations de réglage au début d'une séquence de jobs, d'où l'utilisation du concept du "batching" qui consiste à regrouper les jobs puis à les ordonnancer par groupes. Le temps de réglages sont soit dépendants ou indépendant de la séquence de jobs. Plusieurs fonctions objectifs sont considérées ; elles sont toutes basées sur les temps de fin de traitement et/ou les dates d'échéance des jobs. L'auteur a prouvé que la majorité de ces problèmes est $\mathscr{NP}$-difficile. Cependant, il parvient à établir plusieurs relations de dominance pour certains de ces problèmes. Par exemple, nous retenons que les solutions semi-actives sont dominantes. Par conséquent, lors da la recherche d'une solution optimale, il se restreint à cet ensemble particulier de solutions.

D'autres études furent conduites pour la minimisation de la somme des temps de fin de traitement, ou ce qui est plus communément appelé flowtime. On cite pour l'exemple le papier de [Li et al. (1995)] où les auteurs ont étudié un flow shop à $m$ machines et un opérateur. Ils présentent et comparent trois formulations mathématiques et un algorithme de colonie de fourmis. Les auteurs affirment que cet algorithme est efficace en termes de temps de calcul et qu'il fournit de bons résultats sur le benchmark considéré.

Étant donnée la grande complexité des problèmes d'ordonnancement avec opérateurs, plusieurs études ont traité des problèmes minimaux comme le problème à deux machines. Pour plus de détails, voir e.g. [Baki (1999); Baptiste et al. (2005); Baptiste et Munier (2013); Oulamara et al. (2013)].

D'autres études ont traité des modèles plus réalistes en considérant les temps de transport. On cite pour l'exemple l'article de [Agrawal (2011)] qui a traité un problème de flow shop à 5 machines avec un opérateur et des temps de transport ; l'objectif était la minimisation du makespan. Une formulation mathématique a été présentée pour la résolution de ce problème. L'auteur affirme que cette dernière a permis l'obtention de solutions optimales avec une manière simple et efficace.

Pour finir, nous présentons un exemple d'études faisant intervenir des techniques d'optimisation par contraintes. [Neubert et Savino (2009)] ont étudié un problème de flow shop avec opérateurs dans le contexte d'une ligne d'assemblage avec six postes de travail. Les auteurs ont développé des méthodes de résolution basées sur la satisfaction de contraintes et les techniques d'optimisation par contraintes. Leur approche a surpassé les solutions existantes.

D'autres études ont porté sur des cas pratiques tels que les flow shops flexibles avec des caractéristiques supplémentaires d'opérateurs notamment les phénomènes d'oubli et d'apprentissage [Benavides et al. (2014); Seidgar et al. (2015); Carniel et al. (2015)].

## 3.4 JOB SHOP

Les problèmes de job shop ont été largement étudiés dans la littérature dédiée à l'ordonnancement [Mencia et al. (2015b)]. Ceci est principalement dû à deux raisons ; la première, c'est qu'ils permettent de modéliser un grand nombre de situations rencontrées dans la vie

courante, ce qui les rend très utiles. La seconde, c'est qu'ils comptent parmi les problèmes d'optimisation combinatoire les plus difficiles, leur étude permettrait ainsi de fournir des résultats utiles pour l'avancement de la recherche.

Dans ce qui suit, nous présentons une revue de la littérature dédiée aux problèmes de job shop avec opérateurs.

Les problèmes d'ordonnancement avec opérateurs en général et plus particulièrement dans les environnements de type job shop étant relativement récents et difficiles à résoudre, nous avons remarqué que relativement peu d'études leur ont été consacrées. La difficulté de ces modèles concerne principalement la modélisation des différentes caractéristiques des ressources humaines, telles que l'expérience d'un opérateur, ses aptitudes, la charge de travail variable qui lui est affectée, etc.

Dans les modèles d'ordonnancement classiques, il est implicitement supposé que le nombre d'opérateurs est égal au nombre de machines. Dans ce cas de figure, un opérateur pourra alors se consacrer exclusivement à faire fonctionner la machine qui lui est assignée. Cependant, cette hypothèse n'est que rarement vérifiée en pratique où le nombre d'opérateurs est en général réduit par rapport au nombre de machines.

Étant donnée la difficulté inhérente du problème du job shop, relativement peu de travaux ont porté sur sa version intégrant des ressources humaines. Cependant, lesdits travaux ont fourni des méthodes de résolution compétitives qui fournissaient des résultats de bonne qualité.

Le problème de job shop avec opérateurs avec un nombre d'opérateurs inférieur au nombre de machines a d'abord été introduit dans [Agnetis et al. (2011)]. Les auteurs y fournissent des résultats de complexité pour des cas minimaux et conçoivent des méthodes de résolution efficaces dont un algorithme de programmation dynamique pseudo-polynomial, un schéma

FPTAS (fully polynomial time approximation algorithm), un schéma d'énumération basé sur un graphe disjonctif généralisé qui fut intégré dans une procédure de branch-and-bound et une heuristique basée sur un algorithme de programmation dynamique.

L'algorithme de programmation dynamique a surpassé de loin le branch-and-bound et l'heuristique a fourni des résultats d'assez bonne qualité tout en démontrant de la robustesse.

Nous mentionnons également l'algorithme génétique [Mencia et al. (2011)]. Les auteurs y ont développé un schéma de génération d'ordonnancement inspiré de celui de [Giffler et Thompson (1960)]. Dans cet algorithme, ils affectent des jobs aux opérateurs pour construire une solution valide. Leur schéma garantit l'obtention d'une solution optimale. Ils l'ont utilisé comme décodeur dans le corps d'un algorithme génétique assez classique qui utilise l'opérateur de croisement "order crossover" [Bierwirth (1995)]. Comme la recherche concernant ce problème en était à ses débuts, la seule méthode disponible à laquelle ils ont pu se comparer était celle d'[Agnetis et al. (2011)]. Ils soutiennent que leur algorithme génétique a fourni des résultats compétitifs. De plus, étant donné la rapidité de ce dernier, il y aurait beaucoup de marge d'amélioration. L'algorithme génétique en question fut par la suite amélioré par l'intégration d'une faible évolution Lamarckienne et une technique permettant de réduire l'espace de recherche dans [Mencia et al. (2014)].

Agnetis *et al.* ont aussi proposé deux heuristiques et une formulation mathématique à variables mixtes pour un problème de job shop avec opérateurs qui fut utilisé pour modéliser un cas de production artisanale [Agnetis et al. (2014)]. Leur étude a montré qu'ils parvenaient à obtenir des solutions proches de l'optimale en exploitant les caractéristiques du problème dans leurs heuristiques.

Sierra *et al.* ont développé d'autres schémas de génération d'ordonnancement qui peuvent être utilisés dans différents cadres comme des méthodes heuristiques, des stratégies de branchement

ou en tant que décodeurs dans des algorithmes évolutionnaires. Ils ont amélioré leur précédent schéma [Mencia et al. (2011)] en le rendant plus rapide et plus efficace. Ils utilisent l'algorithme résultant dans une approche exacte qui inclut une stérilisation par des règles de dominance [Sierra et al. (2015)].

Enfin, nous mentionnons les algorithmes mimétiques de [Mencia et al. (2015b)]. Les auteurs les ont conçus avec deux différents constructeurs d'ordonnancement et y ont intégré une méthode de recherche locale basé sur une recherche taboue. Leur structure de voisinage est basée sur l'inversement d'arcs dans un graphe disjonctif. Leurs algorithmes ont été testés sur un grand benchmark et ont produit des résultats de bonne qualité en un temps d'exécution assez court.

La majorité des travaux dédiés au job shop avec opérateurs ont traité l'objectif du makespan. D'autres travaux ont porté sur la minimisation du flowtime ; pour plus de détails, voir [Sierra et al. (2011); Mencia et al. (2013, 2015a)]. D'autres encore ont traité des modèles de job shop flexibles sur des cas réels avec des caractéristiques supplémentaires des opérateurs comme le phénomène d'apprentissage et de fatigue, pour plus de détails, voir [Paksi et Ma'Ruf (2016); Morinaga et al. (2017); Borreguero-Sanchidrian et al. (2018); Dhiflaoui et al. (2018)].

## 3.5 OPEN SHOP

Dans cette section, nous présentons une brève revue de la littérature concernant les problèmes d'open shop et plus particulièrement ceux ayant des contraintes d'opérateurs.

De nombreux travaux ont été consacrés aux problèmes d'open shop durant les dernières décennies. Ceci est notamment dû à leur grande utilité du fait de leurs nombreuses applications en pratique. Les études diverses et variées ont porté sur des problèmes minimaux à deux

machines [Jurisch et Kubiak (1997); Mashuda et Ishii (1994)], des problèmes difficiles pour lesquels ont été établies des formulations mathématiques [Kis et al. (2010)] conçues des méthodes exactes [Brucker et al. (1997); Gueret et Prins (1999)], des méthodes heuristiques [Jurisch et Kubiak (1997); Lu et Posner (1993)] ou encore des métaheuristiques de trajectoire [Liaw (1999); Roshanaei et al. (2010)], de population [Chernykh et al. (2013); Matta (2009)] ou encore des hybrides [Fang et al. (1994); Panahi et Tavakkoli-Moghaddam (1994)]. Même si la grande majorité des travaux ont porté sur l'objectif du makespan, certains articles ont été dédiés à la somme des retards [Liu et Bulfin (1988)] ou encore à la somme des temps de fin de traitement [Sha et al. (2010)]. Pour plus de détails, voir [Anand et Panneerselvam (2015)].

Nous avons noté que relativement peu de travaux ont porté sur les open shops avec opérateurs. Ceci est notamment dû à leur complexité inhérente ainsi qu'à l'apparition récente du domaine d'ordonnancement avec opérateurs.

Les modèles de Vickson [Vickson (1980a,b)] avec temps de traitement contrôlables par la quantité de ressources assignées et les modèles introduisant le concept de partage d'opérateurs dans lesquels le nombre d'opérateurs était inférieur à celui des machines [Baptiste et Munier (2013); Cheurfa (2005); Zouba (2009)] ont principalement été utilisés pour d'autres types d'ateliers tels que le flow shop et le job shop. Il en est de même pour les modèles à un seul opérateur [Baki (1999); Cheng et al. (1999)].

Parmi les rares travaux dédiés aux problèmes d'open shops avec opérateurs, certains ont porté sur des problèmes industriels [G. Campos Ciro (2014)] alors que d'autres modélisaient les opérateurs comme des ressources intervenant seulement au début des tâches [Oulamara et al. (2013)]. Chacun de ces travaux avait pour objectif de réduire l'écart entre théorie et pratique en proposant des modèles qui se voulaient plus réalistes.

# CHAPITRE 4

# FLOW SHOP DE PERMUTATION AVEC TEMPS DE RÉGLAGES

## 4.1 INTRODUCTION

Dans ce chapitre, nous traitons le problème de flow shop de permutation avec temps de réglages dépendants de la séquence de tâches. Les résultats obtenus ont été publiés dans [Benkalai et al. (2016c, 2017d)]. Ce problème peut être décrit comme suit : Soit un ensemble de $n$ jobs indépendant qui doivent être traités par un ensemble de $m$ machines. Chaque job $j$ est composé de $m$ opérations, $O_{1j}$, ... $O_{mj}$, qui doivent être exécutées sans interruption, respectivement dans cet ordre, par la machine $M_1$, la machine $M_2$ et ainsi de suite jusqu'à sa complétion sur la machine $M_m$.

La séquence de jobs est identique sur toutes les machines, *i.e.* si un job est à la $j$ième position sur la machine $M_1$, alors il sera à la $j$ième position sur toutes les autres machines. De plus, si les jobs $j$ et $k$ sont exécutés dans cet ordre sur la machine $M_i$, alors un temps de réglage $s_{ijk}$ devra s'écouler entre la complétion de l'opération $O_{ij}$ et le début de l'opération $O_{ik}$. Ce temps est nécessaire pour l'ajustement des machines avant le début du prochain job de la séquence. Nous supposons que nous avons $m$ opérateurs et que chacun sera affecté à une machine de laquelle il s'occupera exclusivement. Ces derniers se chargeront des opérations de réglage. Le

but est de trouver un ordonnancement réalisable qui minimise le temps de complétion global, aussi appelé makespan. En utilisant la notation d'ordonnancement classique (voir e.g. [Pinedo (2002)]), ce problème est noté $F_m|prmu, s_{ijk}|C_{max}$.

Le problème de flow shop a été largement étudié dans la littérature dédiée à l'ordonnancement. Des progrès considérables ont été accomplis en termes d'efficacité et de rapidité depuis l'article séminal de Johnson en 1954 [Gupta et Stafford (2006)]. Le problème de flow shop de base permet de modéliser plusieurs problèmes réels dans différents domaines tels que les systèmes de production, l'informatique, la gestion de projet et bien d'autres. Cependant, certains problèmes nécessitent la considération de plusieurs paramètres supplémentaires tels que les temps de réglages, les pannes, les retards, etc. La prise en compte de ces derniers engendre généralement des problèmes plus complexes, même pour les cas à une machine (voir e.g. [Pinedo (2002)]). Dans le but d'obtenir des modèles plus réalistes et ainsi de meilleurs outils d'optimisation et d'aide à la décision, il demeure néanmoins nécessaire de considérer des paramètres additionnels au modèle de base. Dans notre cas, nous supposons que ce sont les opérateurs qui effectuent les réglages nécessaires.

L'ordonnancement dans des environnements de type flow shop de permutation avec temps de réglages permet de modéliser nombre de problèmes réels faisant intervenir des machines polyvalentes. Ainsi, plusieurs applications ont été décrites au fil des études portant sur ce problème ; les temps de réglages y étaient nécessaires pour par exemple ajuster une température ou changer un moule de fabrication. On cite pour l'exemple l'industrie du papier où l'on doit ajuster les machines quand on passe d'un lot de coupe à un autre [Ruiz et Stutzle (2005)]. On pourrait trouver une autre application dans les usines de fabrication de pneus. Entre les étapes de fabrication de différents types de pneus, on change les moules de fabrication. Un dernier exemple est celui qu'on rencontre dans les boulangeries par exemple. On peut avoir à effectuer des ajustements entre les cuissons de produits qui nécessitent de températures différentes.

Le problème de flow shop de permutation avec temps de réglages est $\mathscr{NP}$-difficile au sens fort [Gupta (1986)], même pour le cas à une machine [Ruiz et al. (2004)] (ce qui revient à un problème à machine unique). L'utilisation d'une approche heuristique pour la résolution est donc justifiée. Nous utiliserons pour cela une métaheuristique connue sous le nom de Migrating Birds Optimization (MBO) ou optimisation par vol d'oiseaux migrateurs qui a été récemment proposée par Duman et al. [Duman et al. (2012)]. Ce choix est motivé par deux raisons. Tout d'abord, la méthode est simple à assimiler et à implémenter. Ensuite, elle a fourni des résultats prometteurs pour le problème d'affectation quadratique et le problème de flow shop de permutation [Pérez-González et al. (2009); Tongur et Ulker (2014)] ; il serait donc intéressant d'évaluer sa performance sur le problème de flow shop de permutation avec temps de réglages.

## 4.2 DESCRIPTION DU PROBLÈME

Le flow shop de permutation avec réglages (FSPR) peut être décrit comme suit. Soit $J = \{1,2,\ldots,n\}$ un ensemble de $n$ jobs à traiter sans interruption sur un ensemble $M = \{M_1,\ldots,M_m\}$ de $m$ machines. Un temps de traitement $p_{ij}$ représente le temps que passe le job $j$ sur la machine $M_i$, avec un temps de réglage $s_{ijk}$ qui représente le temps qui doit s'écouler entre la fin du job $j$ et le début du job $k$ quand le premier précède le second sur la machine $M_i$ dans une certaine séquence. De plus, la séquence de jobs est identique sur toutes les machines.

Le but est de trouver un ordonnancement valide qui minimise le makespan. En d'autres termes, nous cherchons une séquence $\pi = (\pi(1),\ldots,\pi(n))$ qui contient chaque job exactement une fois de façon à ce que

$$C_{max}(\pi^*) = \min_{\pi\in\Pi} C_{max}(\pi),$$

où $\Pi$ représente l'ensemble des $n!$ permutations possibles, $\pi$ et $\pi^*$ sont des séquences dans $\Pi$ , et $C_{max}(\pi)$ est le temps de complétion final du dernier job de la séquence $\pi$.

La valeur du makespan pour une séquence donnée $\pi$ est calculée comme suit. D'abord, soit $C(\pi(j),i)$ le temps de fin du job $\pi(j)$ sur la machine $M_i$. Le makespan d'une séquence donnée $\pi$ est égal au temps de fin du dernier job de cette dernière (le job à la $n$ième position), sur la machine $M_m$, *i.e.* $C_{max}(\pi) = C(\pi(n),m)$. Étant donnée une solution valide $\pi$, les temps de fin de traitement des $n$ jobs sur les $m$ machines, $C(\pi(j),k)$, $1 \leq j \leq n$ et $1 \leq k \leq m$, sont calculés suivant la formule récursive suivante.

$$C(\pi(1),k) = \sum_{i=1}^{k} p_{i,\pi(1)}; k = 1,\ldots,m, \tag{4.1}$$

$$C(\pi(j),1) = \sum_{q=1}^{j-1} (p_{1,\pi(q)} + s_{1,\pi(q),\pi(q+1)}) + p_{1,\pi(j)}; j = 2,\ldots,n, \tag{4.2}$$

$$C(\pi(j),k) = \max\{C_{k-1,\pi(j)}; C_{k,\pi(j-1)} + s_{k,\pi(j-1),\pi(j)}\} + p_{k,\pi(j)}; \tag{4.3}$$

$$k = 2,\ldots,m;\ j = 2,\ldots,n.$$

Les équations (4.1) calculent les temps de fin de traitement du premier job de la séquence $\pi$ sur les $m$ machines. Quand il arrive sur la machine $M_k$, $k = 1,\ldots,m$, le premier job a déjà été traité sans interruption par les $k-1$ premières machines. De ce fait, $C(\pi(1),k)$, $k = 1,\ldots,m$, est égal à la somme des temps de traitement de $\pi(1)$ sur les $k$ premières machines. Lors de son passage sur la machine $M_k$, $k = 1,\ldots,m$, le premier job est traité par l'opérateur affecté à la machine en question.

Les équations (4.2) calculent les temps de fin de traitement des $(n-1)$ derniers jobs sur la première machine. En effet, les $n$ jobs sont traités sans interruption sur la première machine

par l'opérateur qui lui est affecté. Ainsi, $C(\pi(j),1)$, $j = 2,\ldots,n$, est égal à la somme des temps de traitement des jobs précédant $\pi(j)$ dans la séquence $\pi$ plus la somme des temps de réglages nécessaires plus $p_{\pi(j),1}$.

Finalement, les équations (4.3) expliquent comment calculer les temps de fin de traitement des jobs restants. Un job $\pi(j)$, $j = 2,\ldots,n$, commence sur la machine $M_k$, $k = 2,\ldots,m$, après avoir terminé son traitement sur la machine $M_{k-1}$, après que le job $\pi(j-1)$ ait terminé son traitement sur la machine $M_k$, et que le temps de réglage nécessaire $s_{k,\pi(j-1),j}$ se soit écoulé.

Bien sûr, le fait qu'aucune machine ne reste inactive alors qu'elle a un job en attente vient du fait que le nombre d'opérateurs est égal au nombre de machine et que donc chaque opérateur peut se consacrer exclusivement à la seule machine à laquelle il est affecté.

Par soucis de clarté, nous présentons l'exemple suivant sur une instance $I$ du problème FSPR avec 3 machines et 3 jobs. Les temps de traitement $p_{ij}$ et les temps de réglage $s_{ijk}$ sont présentés en Table 4.1. Le diagramme de Gantt de la solution $\pi = (1,2,3)$ est illustré à la Figure 4.1. La valeur du makespan de la solution est $C_{max}(\pi) = 10.5$.

| | $p_{i1}$ | $p_{i2}$ | $p_{i3}$ | $s_{i12}$ | $s_{i13}$ | $s_{i21}$ | $s_{i23}$ | $s_{i31}$ | $s_{i32}$ |
|---|---|---|---|---|---|---|---|---|---|
| $i = 1$ | 2 | 1.5 | 1.8 | 0.5 | 1 | 0.4 | 0.2 | 0.7 | 2 |
| $i = 2$ | 2 | 1.5 | 1.4 | 0.5 | 1.4 | 2 | 0.1 | 0.5 | 0.8 |
| $i = 3$ | 1.5 | 3.5 | 0.4 | 1 | 1.5 | 0.2 | 0 | 0.3 | 0.1 |

**Tableau 4.1: Données de l'instance *I*.**

## 4.3 L'APPROCHE MBO DE BASE

La méthode MBO est une métaheuristique à population de solutions. Son principe de base consiste en une exploration structurée des voisinages des individus de la population. Dans ce qui suit, nous décrivons la méthode ainsi que son analogie avec le phénomène naturel de la

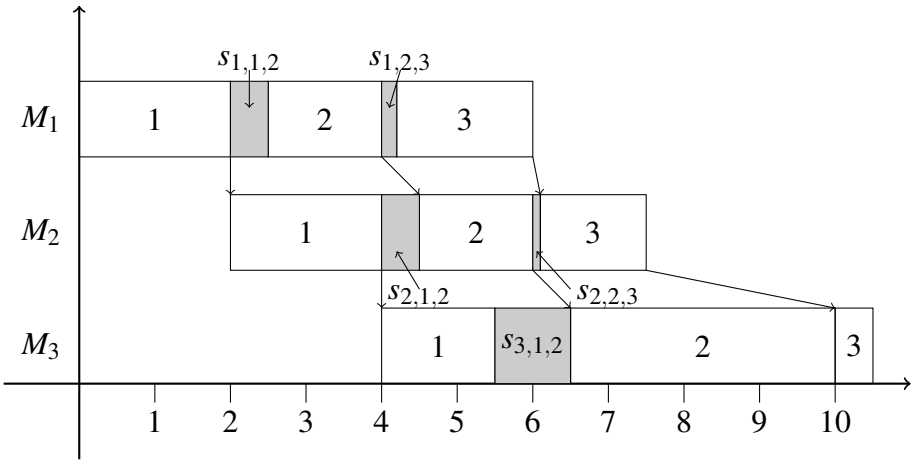



**Figure 4.1: Le diagramme de Gantt et le makespan de la solution $\pi = (1,2,3)$.**

migration des oiseaux.

### *4.3.1 L'ALGORITHME*

La méthode MBO est une métaheuristique inspirée du vol d'oiseaux migrateurs, précisément de leur formation de vol en "V" [1] qui est connue pour ses propriétés de préservation d'énergie.

Pour un oiseau seul, le facteur le plus important lors de son vol est la *vitesse d'avancement* (forward speed) : plus elle est élevée, plus il s'élève. La force nécessaire pour générer cette élévation est appelée *force induite* (induced power).

Quand des oiseaux volent en groupe dans une formation en "V", quand l'un d'entre eux bat des ailes, l'air circule au-dessus et en dessous de chaque aile générant ainsi deux vortex tubulaires qui contribuent à l'élévation du l'oiseau suivant, ainsi réduisant son besoin en force induite ainsi que l'énergie qu'il dépense [Duman et al. (2012)].

Cette économie d'énergie dépend de plusieurs paramètres tels que la distance entre les oiseaux

1. L'appellation vient de la similarité avec la lettre "V".

et leur nombre. Nous remarquons que l'oiseau de tête (le leader) est celui qui dépense le plus d'énergie. Après un certain temps, il ira se positionner à l'arrière du groupe en laissant sa place à l'un des deux oiseaux qui le suivaient.

Avant de décrire l'algorithme de la métaheuristique MBO en Figure 4.2, nous allons lister les paramètres de cette approche :

— $\ell$ : nombre de solutions.

— $k$ : nombre de voisins à évaluer pour chaque solution.

— $x$ : nombre de voisins partagés avec la solution suivante.

— $t$ : nombre d'itérations avant le changement de la solution de tête.

— $K$ : nombre maximum d'itérations.

```
Générer ℓ solutions aléatoirement ;
Placer les solutions arbitrairement de façon à former une forme en "V" ;
i ← 0 ;
Tant que(i < K)
{
    Pour(j = 0; j < t; j++)
    {
        Améliorer la solution de tête en générant k voisins ;
        i ← i + k ;
        Pour(Chaque solution s_r du vol (exceptée celle associée au leader))
        {
            Améliorer s_r en évaluant (k − x) de ses voisins ainsi que x voisins
            non-utilisés de la solution la devançant ;
            i ← i + (k − x) ;
        }
    }
    Changer le leader ;
}
Retourner la meilleure solution du vol ;
```

**Figure 4.2: Pseudo-code de la méthode MBO [Duman et al. (2012)].**

Après avoir généré aléatoirement $\ell$ solutions et les avoir placées en forme de "V" [2], l'algorithme commence par tenter d'améliorer la solution leader en générant $k$ de ses voisins. Le meilleur d'entre eux remplacera éventuellement ladite solution s'il est meilleur qu'elle. Ensuite, le mécanisme de partage propre à la méthode entre en jeu. Pour chaque solution, autre que celle de tête, on génère seulement $(k-x)$ solutions et l'on prend les $x$ autres de l'ensemble des voisins non-utilisés de la solutions la devançant. La meilleure de ces $(k-x)+x$ solutions remplacera alors la solution considérée si elle est meilleure qu'elle. Sinon, la population reste inchangée.

Ces étapes sont répétées pendant $t$ itérations, ensuite la solutions de tête est changée et remplacée par la solution derrière elle à sa droite. L'algorithme s'arrête après avoir généré un nombre donné de $K$ voisins.

Pour calculer la complexité de l'algorithme, nous procédons comme suit. D'abord, notons que la génération des premières $\ell$ solutions se fait en $O(\ell n^2)$. La génération d'un voisin se fait en $O(1)$. Le tri de la liste des voisins peut être implémenté en $O(k \log k)$. L'affectation d'une solution à une autre se fait en $O(n)$. Finalement, l'évaluation d'une solution se fait en $O(mn)$. Comme la boucle *tant que* se répète au plus $O(K)$ fois, le temps d'exécution de l'algorithme est en $O(\ell n^2 + K(t\ell(k(mn + k\log(k)))))$.

### *4.3.2 ANALOGIE*

La Table 4.2 présente l'analogie entre l'algorithme MBO et le phénomène naturel de la migration des oiseaux.

2. Nous utilisons une structure de données qui garde en mémoire les positions du prédécesseur et successeur de chaque solution.

| MBO | Vol d'oiseaux migrateurs |
|---|---|
| Nombre de solutions $\ell$ | Nombre d'oiseaux |
| Nombre de voisins évalués $k$ | Force induite requise* |
| Nombre de voisins partagés $x$ | Distance entre les oiseaux |
| Nombre d'itérations $t$ | Temps de vol d'un oiseau comme leader |

*Force nécessaire à un oiseau pour s'élever.

**Tableau 4.2: L'analogie entre la méthode MBO et le phénomène naturel de migration.**

## 4.4 ÉTUDE EXPÉRIMENTALE

Dans ce qui suit, nous présentons deux versions de l'approche MBO. Tout d'abord, nous avons adapté la version de base, telle que décrite dans [Duman et al. (2012)]. Nous avons par la suite proposé une version améliorée.

Les deux algorithmes ont été codés en C++, et débogués sous Microsoft Visual Studio 2015 sur une workstation Dell avec un processeur Intel $^{TM}$ Xeon$^{TM}$ CPU E5-1620 à 3.70 GHz et une RAM de 16 GB.

Pour commencer, nous présentons notre benchmark. Ensuite, nous décrivons les caractéristiques de nos algorithmes et analysons les résultats obtenus.

### *4.4.1 DESCRIPTION DU BENCHMARK*

Pour tester les algorithmes que nous avons conçus, nous avons choisi un ensemble de 480 instances pour lesquelles les combinaisons possibles pour $n \times m$ sont $\{20,50,100\}$ $\times$ $\{5,10,20\}$, $200 \times \{10,20\}$, et $500 \times 20$. Ces instances, disponibles sur le site `http://soa.iti.es/problem-instances`, sont des extensions des instances de Taillard pour le problème de flow shop de permutation auxquelles des temps de réglages ont été ajoutés ;

pour chaque instance est disponible le meilleur résultat connu. Pour plus de détails, voir [Vallada et al. (2003)]. Le benchmark est divisé en 4 ensembles suivant l'ordre de grandeur des temps de réglages. Ils sont comme suit :

- **SDST10** : Les temps de réglages sont uniformément tirés entre 1 et 9 (10% des temps de traitement).
- **SDST50** : Les temps de réglages sont uniformément tirés entre 1 et 49 (50% des temps de traitement).
- **SDST100** : Les temps de réglages sont uniformément tirés entre 1 et 99 (100% des temps de traitement).
- **SDST125** : Les temps de réglages sont uniformément tirés entre 1 et 124 (125% des temps de traitement).

### *4.4.2 PREMIÈRE VERSION DE L'ALGORITHME*

La première méthode que nous présentons est un MBO de base (BMBO).

Dans ce qui suit, nous décrivons le voisinage choisi ainsi que la phase de réglage des paramètres et nous analysons les résultats obtenus.

#### A. Le voisinage 3-interchange

La transformation 3-interchange consiste à échanger, dans une solution donnée $S$ les positions de trois jobs pour produire une autre solution $S'$. Dans la Figure 4.3, nous présentons un exemple où les jobs échangés sont aux positions 3, 5, et 9.

Le voisinage défini par la transformation sus-mentionnée est choisi pour sa capacité à préserver les positions absolues des jobs dans une solution.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $S$ | 1 | 2 | **3** | 4 | **5** | 6 | 7 | 8 | **9** |
| $S'$ | 1 | 2 | **9** | 4 | **3** | 6 | 7 | 8 | **5** |

**Figure 4.3: La transformation 3-interchange.**

**B. Réglage des paramètres**

La première version de l'algorithme a été testée seulement sur un ensemble réduit d'instances de cardinalité 16. L'ensemble contenait des instances de quatre tailles différentes, à savoir, $20 \times 5, 20 \times 10, 20 \times 20$ et $50 \times 5$, et une instance par taille par ordre de grandeur de temps de réglage (les instances $ta001, ta011, ta021$ et $ta031$).

Les solutions sont représentées par des vecteurs de taille $n$. La population initiale a été générée aléatoirement en utilisant une heuristique constructive qui ordonnance à chaque itération un job $j$, $j = 1, \ldots, n$, à une position aléatoire $h$, $1 \leq h \leq n$, jusqu'à ce que le vecteur soit complété.

Le voisinage a été implémenté de telle manière à toujours générer des solutions réalisables. Ceci est aussi vérifié pour les algorithmes de la Section 4.4.3.

Ayant fixé $K$ à 10000 comme étant le nombre maximum d'évaluations de solutions, nous avions quatre autres paramètres à régler.

Tout d'abord, en nous inspirant des directives données par rapport aux valeurs des paramètres dans [Duman et al. (2012)], nous avons fixé des valeurs de références aux dits paramètres comme suit : $\ell = 21$, $k = 5$, $t = 2$, et $x = 2$. Ensuite, nous les avons réglés, un à la fois. La Figure 4.4 présente les pourcentages de déviation moyens (l'axe $Oy$) obtenus pour chaque groupe d'instances (l'axe $Ox$) pour chaque étape de réglage des paramètres. L'axe $Ox$ représente les groupes d'instances sous le format : nombre de jobs $\times$ nombre de machines.

Les pourcentages de déviation $d$ ont été calculés en utilisant la formule suivante

$$d = \frac{100(Z_{BMBO} - Z^*)}{Z^*}, \tag{4.4}$$

où $Z_{BMBO}$ et $Z^*$ sont l'évaluation de la solution générée par *BMBO* et la meilleure solution connue, respectivement.

**Le nombre d'oiseaux :**

Dans la méthode MBO, il est recommandé de prendre des valeurs impaires pour le paramètre $\ell$ [Duman et al. (2012)]. En effet, en plus de l'oiseau de tête, nous voudrions idéalement avoir le même nombre d'oiseaux sur les deux branches du "V". Dans [Duman et al. (2012)], les auteurs recommandent des tailles de vol relativement petites. Suivant cette recommandation, nous avons testé *BMBO* sur des tailles de vol allant de 11 à 41.

Nous avons exécuté la méthode 20 fois pour chaque instance

La Figure 4.4a présente les résultats obtenus pour chaque groupe d'instances et pour chaque valeur testée pour le paramètre $\ell$, à savoir, 11, 21, 31 et 41. Les résultats nous ont poussés à fixé la taille du vol à 11, cette taille ayant donné les meilleurs résultats moyens.

**Le nombre de voisins évalués :**

Avec le nombre limité de 10000 évaluations possibles, nous avons testé *BMBO* pour de petites valeurs de $k$ pour permettre l'exploration des voisinages d'un plus grand nombre de solutions pendant le déroulement de l'algorithme. La Figure 4.4b présentes le pourcentage de déviation moyen pour chaque groupe d'instances et pour chaque valeur de $k$. La méthode a été testée

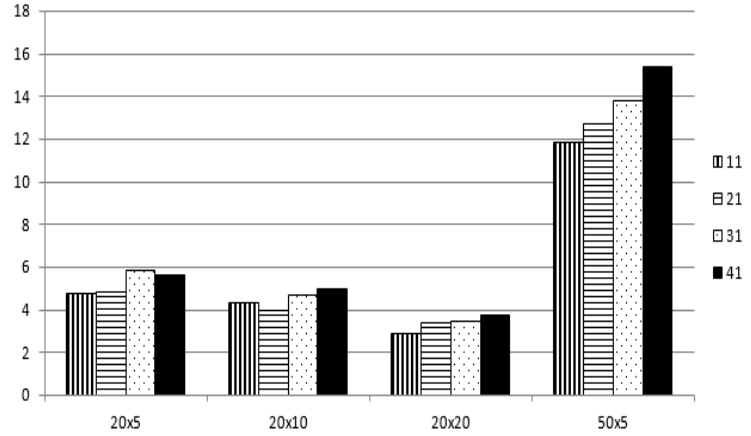
pour $k = 5,\ 10,\ 15,\ 20$ ; les résultats montrent que $k = 5$ donne les meilleurs résultats moyens, cette valeur a donc été gardée pour la suite des tests.

**Le temps passé par une solution en tant que leader :**

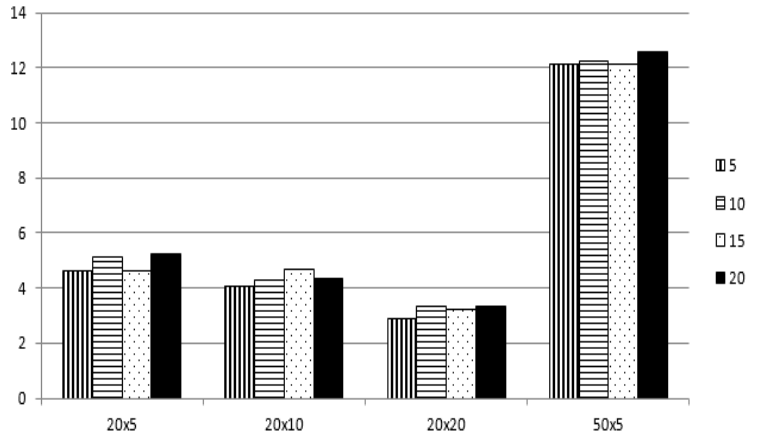
Le changement de leader permet une meilleure exploration des voisinages des nouvelles solutions. Ici encore, il est recommandé de prendre de petites valeurs pour la paramètre $t$. À la lumière des résultats obtenus pour les précédents paramètres et étant donné le nombre $K$, nous avons exécuté la méthode 20 fois sur chaque instance pour des valeurs de $t$ allant de 1 à 4. Les résultats obtenus sont illustrés par la Figure 4.4c ; il nous ont fait choisir la valeur $t = 1$.

**Le nombre de voisins partagés :**

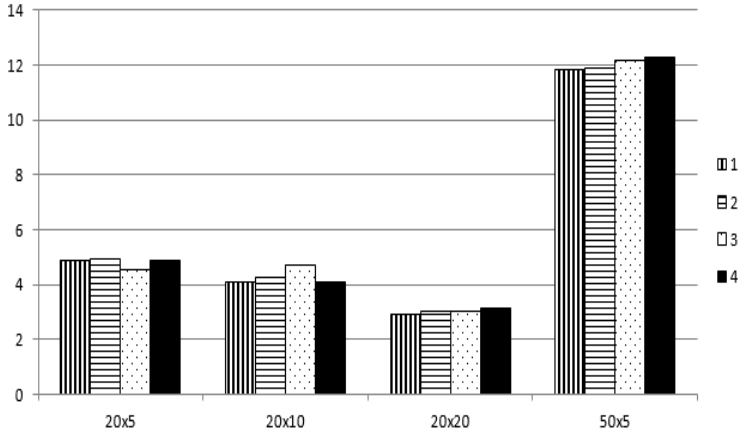
Dans [Duman et al. (2012)], il est suggéré de prendre de petites valeurs pour le paramètre $x$ avec $x \leq k-1$. Dans la série de 20 exécutions effectuée sur notre benchmark, nous avons testé les valeurs $1, 2, 3$ et 4 pour $x$. La Figure 4.4d illustre les pourcentages de déviation moyens obtenus. À la lumière de ceux-ci, nous avons conclu qu'il était plus adéquat de fixer $x$ à 4.



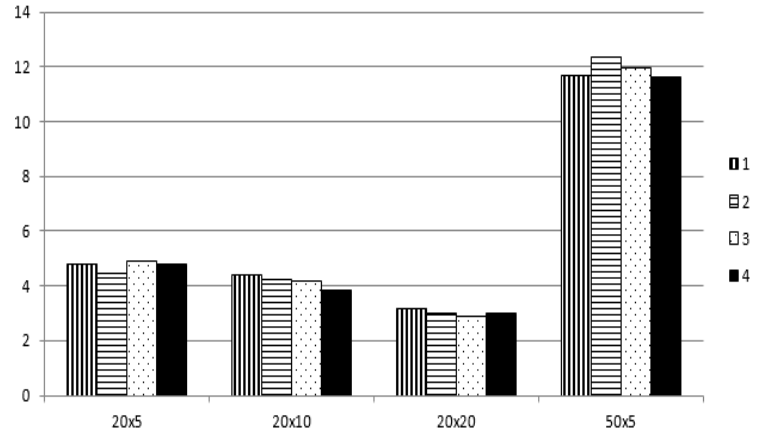

**(a) Effets de la variation de $\ell$ (la taille du vol).**




**(b) Effets de la variation de $k$ (la force induite requise).**




**(c) Effets de la variation de $t$ (temps de vol du leader).**




**(d) Effets de la variation de $x$ (distance entre les oiseaux).**

**Figure 4.4: Réglage des paramètres de *BMBO*.**

### C. Analyse des résultats numériques

L'algorithme *BMBO* a été exécuté cinq fois sur chaque instance. Les résultats obtenus sont présentés en Table 4.3. Notons que tous les résultats calculés (moyenne, écart-type, etc.) le sont en fonction des pourcentages de déviations $d$ relativement à la meilleure solution connue tel que décrit par l'équation (4.4).

Les résultats complémentaires sont comme suit :

écart-type : 4.63, variance : 21.48, moyenne : 5.61, maximum : 17.69, minimum : 1.26.

| SDST10 | | | |
|---|---|---|---|
| ta001 | ta011 | ta021 | ta031 |
| 1,26 | 2,34 | 1,62 | 3,05 |

| SDST50 | | | |
|---|---|---|---|
| ta001 | ta011 | ta021 | ta031 |
| 3,62 | 3,56 | 2,78 | 9,95 |

| SDST100 | | | |
|---|---|---|---|
| ta001 | ta011 | ta021 | ta031 |
| 6,32 | 4,24 | 2,98 | 15,26 |

| SDST125 | | | |
|---|---|---|---|
| ta001 | ta011 | ta021 | ta031 |
| 7,03 | 4,78 | 3,25 | 17,69 |

**Tableau 4.3: Pourcentages de déviation moyens.**

À la lumière des résultats présentés en Table 4.3, nous pouvons conclure que la méthode *BMBO* a produit des résultats de qualité acceptable étant donné le nombre limité d'évaluations de solutions autorisé, qui est assez restrictif pour une métaheuristique. En effet, la méthode parvient à approcher les meilleures solutions avec un pourcentage de déviation de moins de 2% pour certaines instances et ne dépasse pas 18% pour les instances de plus grande taille.

Nous remarquons que l'efficacité de la méthode n'est pas vraiment affectée par les variations en temps de réglage ni par l'augmentation du nombre de machines. Elle est cependant affectée par l'augmentation du nombre de jobs. Pour améliorer la qualité de l'algorithme, il serait ainsi intéressant d'expérimenter d'autres structures de voisinage.

Si nous considérons seulement les plus grandes instances avec 50 jobs et 5 machines, nous remarquons que le pourcentage de déviation moyen est de 11.49% et l'écart-type est 5.61. En utilisant les résultats précédents, nous observons que la moyenne pourrait être réduite à 8.5% avec un seuil de signification de $\alpha$=1%. De plus, la moyenne de tous les résultats pourrait atteindre 4.3% au même seuil de signification (l'intervalle de confiance de niveau 99% étant $[4.27; 6.94]$).

Dans ce qui suit, nous présentons une version améliorée de cet algorithme.

### *4.4.3 UNE VERSION AMÉLIORÉE DE LA MÉTHODE MBO*

Comme la qualité des résultats de la méthode *BMBO* a continué à diminuer avec l'augmentation de la taille des instances, nous n'en avons pas plus exploré le comportement mais avons plutôt décidé de l'améliorer. Dans ce qui suit, nous présentons le voisinage et les heuristiques utilisés dans la méthode améliorée que nous notons *EMBO* (Enhanced MBO) ainsi que l'analyse des résultats fournis par cette dernière.

#### A. Adaptation du voisinage Or-Opt

Tel que mentionné précédemment, nous avions choisi le premier voisinage (3-interchange) pour sa capacité à préserver les positions absolues des jobs dans une séquence. Nous avons

remarqué par la suite qu'il était aussi intéressant de préserver des bonnes séquences de jobs en raison de la présence de temps de réglages. Dans ce qui suit, nous présentons un voisinage qui allie ces deux caractéristiques.

Le voisinage Or-opt a été d'abord conçu pour le problème du voyageur de commerce [Or (1976)]. Il consiste à déplacer un bloc d'un, deux ou trois jobs. Il est illustré en Figure 4.5. Nous l'avons modifié pour permettre le déplacement de blocs de quatre jobs et avons nommé la transformation résultante 4*Or*-opt. En effet, quelques résultats préliminaires nous avaient permis de voir que l'ajout de cette possibilité améliorait la diversification de l'algorithme résultant car cela permettait d'atteindre un ensemble plus large de solutions.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $S$ | 1 | 2 | 3 | **4** | **5** | 6 | 7 | 8 | 9 |
| $S'$ | 1 | 2 | 3 | 6 | 7 | 8 | **4** | **5** | 9 |

**Figure 4.5: La transformation $40r$-opt pour un déplacement d'un bloc de deux jobs.**

### B. Heuristiques

Pour améliorer encore plus l'efficacité de notre algorithme, au lieu de démarrer l'exécution avec un ensemble de solutions aléatoires, nous avons inséré dans la formation de départ une solution fournie par une heuristique connue pour donner de bons résultats pour le problème FSPR. Quatre algorithmes, $EBMO_i$, $i = 1,\ldots,4$, ont ainsi été conçus, chacun utilisant l'heuristique $H_i$, $i = 1,\ldots,4$, pour générer son oiseau de tête. Les quatre heuristiques sont comme suit :

$H_1$ : L'heuristique NEH-RMB [Rìos-Mercado et Bard (1998)]. C'est une adaptation de l'heuristique NEH [Nawaz et al. (1983)]. Elle ordonne les jobs dans l'ordre décroissant de la somme de leurs temps de traitement respectifs $\sum_{i=1}^{m} p_{ij}$, ensuite, elle insère de façon séquentielle chaque job, suivant cet ordre, à la position donnant le makespan minimum

tout en prenant en considération les temps de réglage.

$H_2$ : L'heuristique NEH-RMB-AvgStd [Dong et al. (2009)]. Elle est similaire à la précédente. La différence réside dans le fait que celle-ci ordonne les jobs selon l'ordre décroissant de la moyenne de leurs temps de traitement ($Avg_j$) respectifs plus l'écart-type de ces derniers ($Std_j$). $Avg_j + Std_j = \frac{1}{m}\sum_{i=1}^{m} p_{ij} + \left[\frac{1}{m-1}\sum_{i=1}^{m}(p_{ij} - Avg_j)^2\right]^{\frac{1}{2}}$.

$H_3$ : L'heuristique NEH-RMB-AvgStd+ST [Dong et al. (2009)]. Celle-ci est aussi similaire à la première, à la différence qu'elle ordonne les jobs selon l'ordre décroissant des $Avg_j + Std_j + ST_j$, $ST_j$ étant la moyenne des temps de réglages du job $j$.

$H_4$ : L'heuristique NEH-RMB-AvgStd-ST [Dong et al. (2009)]. Encore une fois similaire aux précédentes heuristiques, celle-ci ordonne les jobs selon l'ordre décroissant des $Avg_j + Std_j - ST_j$.

### C. Analyse des résultats numériques

Nous avons testé les algorithmes $EMBO_i$, $i = 1,\ldots,4$, sur le benchmark entier tel que décrit à la Section 4.4.1. Nous avons fixé le nombre maximum d'évaluations de solutions à $K = 10000$. Nos résultats préliminaires ont suggéré de garder les valeurs des paramètres obtenus à la Section 4.4.2. Chaque algorithme a été exécuté trois fois sur chaque instance. Les résultats sont présentés en Table 4.4 ; toutes les valeurs calculées (moyenne, écart-type, variance, etc.) sont en fonction des pourcentages de déviation moyens $d_i$ relativement à la meilleure solution connue :

$$d_i = \frac{100(Z_{EMBO_i} - Z^*)}{Z^*}, i = 1,\ldots,4.$$

où $Z_{EMBO_i}$ et $Z^*$ sont l'évaluation de la solution générée par $EMBO_i$, $i = 1,\ldots,4$, et la meilleure solution connue, respectivement.

| | Méthodes | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SDST10 | | | | SDST50 | | | | SDST100 | | | | SDST125 | | | | |
| | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $T$ |
| 20x5 | 1,55 | 1,33 | 2,67 | 1,00 | 3,14 | 3,73 | 4,43 | 3,72 | 7,34 | 6,28 | 5,14 | 6,20 | 6,77 | 6,05 | 6,98 | 8,16 | 77,06 |
| | 0,77 | 0,61 | 0,69 | 0,60 | 1,80 | 1,77 | 2,24 | 1,85 | 2,91 | 3,47 | 2,93 | 3,05 | 3,28 | 3,02 | 3,35 | 3,52 | |
| | 0,15 | 0,08 | 0,08 | 0,22 | 0,50 | 0,13 | 0,65 | 0,46 | 0,23 | 1,37 | 0,32 | 1,11 | **0,00** | 0,09 | 0,10 | 0,51 | |
| 20x10 | 2,32 | 2,08 | 2,80 | 2,45 | 3,37 | 3,40 | 2,91 | 3,68 | 3,81 | 4,65 | 4,02 | 4,53 | 4,82 | 5,90 | 5,15 | 4,48 | 140,15 |
| | 1,14 | 0,96 | 1,46 | 1,12 | 1,63 | 1,67 | 1,87 | 1,99 | 2,60 | 2,67 | 2,41 | 2,88 | 2,83 | 3,30 | 2,90 | 2,81 | |
| | 0,31 | 0,25 | 0,51 | 0,34 | 0,31 | 0,54 | 0,97 | 0,60 | 0,61 | 0,61 | 0,26 | 1,81 | 0,20 | 1,68 | 1,12 | 0,96 | |
| 20x20 | 2,78 | 2,20 | 1,81 | 2,01 | 2,37 | 2,94 | 2,83 | 2,54 | 3,22 | 3,48 | 3,29 | 3,38 | 3,99 | 3,97 | 4,83 | 3,84 | 268,78 |
| | 1,20 | 0,94 | 0,89 | 0,90 | 1,47 | 1,35 | 1,61 | 1,48 | 1,98 | 2,15 | 2,07 | 2,06 | 2,01 | 2,23 | 2,17 | 1,93 | |
| | 0,39 | 0,13 | **0,00** | 0,21 | 0,33 | 0,08 | 0,71 | 0,11 | 0,51 | 0,87 | 1,00 | 0,91 | 0,89 | 0,17 | 0,41 | 0,49 | |
| 50x5 | 2,81 | 2,45 | 2,36 | 2,60 | 7,52 | 7,82 | 7,88 | 8,73 | 12,50 | 11,66 | 11,83 | 12,79 | 12,66 | 13,37 | 13,08 | 13,89 | 155,04 |
| | 1,78 | 1,84 | 1,79 | 1,82 | 6,39 | 5,99 | 6,19 | 6,23 | 9,75 | 9,20 | 9,44 | 9,28 | 10,50 | 10,61 | 10,77 | 10,67 | |
| | 0,98 | 1,19 | 1,07 | 1,29 | 4,92 | 3,83 | 5,04 | 4,87 | 7,13 | 6,96 | 6,57 | 6,19 | 7,86 | 8,20 | 8,42 | 8,37 | |
| 50x10 | 4,60 | 4,64 | 4,97 | 5,16 | 6,93 | 7,03 | 7,54 | 6,78 | 9,46 | 10,04 | 9,77 | 9,59 | 11,15 | 11,87 | 11,44 | 9,97 | 285,58 |
| | 3,64 | 3,64 | 3,56 | 3,79 | 5,58 | 6,05 | 5,65 | 5,37 | 7,85 | 7,46 | 8,06 | 7,80 | 8,63 | 8,47 | 8,60 | 8,33 | |
| | 2,56 | 2,23 | 2,54 | 2,30 | 4,42 | 4,68 | 3,85 | 4,16 | 5,94 | 5,13 | 6,68 | 6,64 | 6,21 | 6,39 | 7,02 | 6,01 | |
| 50x20 | 5,80 | 4,89 | 5,25 | 4,91 | 6,25 | 6,29 | 6,81 | 6,23 | 7,77 | 7,02 | 7,27 | 7,49 | 8,71 | 7,81 | 8,64 | 7,73 | 544,33 |
| | 3,96 | 4,04 | 4,09 | 4,07 | 5,18 | 4,86 | 4,84 | 4,63 | 6,11 | 5,78 | 6,04 | 5,78 | 6,51 | 6,44 | 6,77 | 6,55 | |
| | 2,87 | 2,85 | 2,89 | 3,18 | 4,07 | 3,99 | 3,65 | 3,70 | 4,64 | 4,66 | 3,79 | 4,50 | 5,15 | 4,68 | 5,48 | 5,24 | |
| 100x5 | 2,72 | 2,69 | 2,91 | 2,54 | 8,46 | 8,79 | 8,56 | 8,34 | 12,74 | 12,02 | 12,40 | 11,38 | 13,50 | 13,91 | 14,04 | 16,05 | 283,71 |
| | 2,18 | 2,18 | 2,20 | 2,18 | 7,35 | 7,14 | 7,02 | 7,33 | 10,92 | 10,65 | 10,93 | 10,20 | 11,89 | 11,92 | 12,15 | 12,46 | |
| | 1,77 | 1,59 | 1,50 | 1,54 | 6,35 | 6,13 | 5,74 | 5,81 | 9,04 | 8,41 | 8,43 | 8,38 | 10,20 | 10,06 | 10,74 | 9,79 | |
| 100x10 | 3,55 | 3,71 | 3,78 | 3,42 | 6,99 | 6,63 | 7,31 | 7,01 | 9,30 | 8,98 | 10,51 | 10,22 | 10,18 | 10,87 | 10,61 | 11,05 | 524,52 |
| | 2,84 | 2,94 | 2,91 | 2,79 | 6,14 | 5,80 | 5,89 | 5,78 | 7,91 | 7,95 | 8,33 | 7,79 | 9,08 | 9,49 | 9,04 | 9,33 | |
| | 2,19 | 2,09 | 2,31 | 2,28 | 5,14 | 4,81 | 4,82 | 4,95 | 6,51 | 6,55 | 7,11 | 6,23 | 7,93 | 8,04 | 7,55 | 8,07 | |
| 100x20 | 4,47 | 4,43 | 4,66 | 4,50 | 5,44 | 6,15 | 5,49 | 5,69 | 7,90 | 7,89 | 7,28 | 8,00 | 7,82 | 7,48 | 7,92 | 7,58 | 1007,50 |
| | 3,50 | 3,62 | 3,88 | 3,78 | 4,87 | 5,11 | 4,91 | 5,04 | 6,48 | 6,69 | 6,32 | 6,85 | 6,47 | 6,63 | 6,85 | 6,44 | |
| | 2,71 | 2,39 | 3,16 | 2,83 | 4,40 | 4,57 | 3,89 | 3,91 | 5,25 | 6,02 | 4,74 | 5,43 | 5,06 | 5,69 | 5,28 | 5,63 | |
| 200x10 | 3,24 | 2,76 | 2,88 | 2,81 | 5,87 | 6,00 | 5,94 | 5,80 | 7,76 | 8,17 | 8,23 | 7,99 | 8,30 | 9,43 | 9,65 | 8,60 | 1031,53 |
| | 2,42 | 2,28 | 2,37 | 2,41 | 5,26 | 5,15 | 5,29 | 5,21 | 6,84 | 7,26 | 7,30 | 6,89 | 7,68 | 7,80 | 7,94 | 7,74 | |
| | 1,97 | 1,67 | 1,93 | 1,82 | 4,61 | 4,50 | 4,23 | 4,20 | 5,12 | 6,35 | 6,65 | 5,48 | 6,78 | 7,02 | 6,42 | 6,80 | |
| 200x20 | 3,89 | 3,85 | 3,57 | 3,58 | 4,61 | 4,36 | 4,45 | 4,87 | 5,77 | 5,86 | 5,70 | 5,59 | 5,84 | 6,42 | 6,39 | 6,71 | 1939,71 |
| | 2,94 | 2,84 | 2,86 | 2,98 | 3,95 | 3,82 | 3,90 | 4,09 | 4,96 | 5,16 | 4,94 | 5,06 | 5,24 | 5,51 | 5,68 | 5,52 | |
| | 2,35 | 2,39 | 2,36 | 2,22 | 3,40 | 3,05 | 3,20 | 3,55 | 3,60 | 4,51 | 4,40 | 4,26 | 4,43 | 4,72 | 4,88 | 4,41 | |
| 500x20 | 1,95 | 1,91 | 2,00 | 2,05 | 2,90 | 2,81 | 2,91 | 3,18 | 3,77 | 3,69 | 3,73 | 4,07 | 4,04 | 4,15 | 3,99 | 4,17 | 5037,66 |
| | 1,73 | 1,69 | 1,70 | 1,65 | 2,66 | 2,50 | 2,67 | 2,64 | 3,36 | 3,35 | 3,30 | 3,40 | 3,64 | 3,65 | 3,57 | 3,52 | |
| | 1,54 | 1,30 | 1,43 | 1,37 | 2,36 | 2,13 | 2,30 | 2,23 | 2,90 | 3,07 | 2,84 | 3,04 | 3,38 | 3,18 | 3,07 | 3,06 | |

**Tableau 4.4: Pourcentage de déviation moyen du makespan des solutions comparé à la meilleure solution connue.**

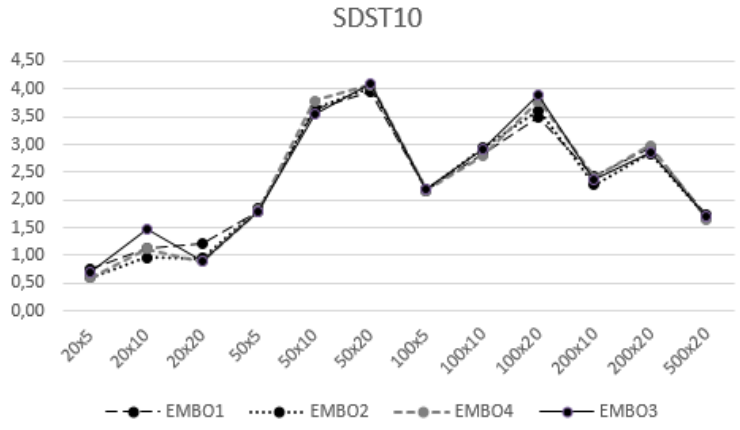
Nous avons remarqué qu'aucune méthode n'était plus rapide que les autres de façon significative, alors nous avons calculé $T$, le temps d'exécution moyen en secondes par instance, par taille d'instance. Pour chaque taille d'instance, nous présentons, dans cet ordre, le maximum, la moyenne et le minimum des pourcentages de déviation. L'écart-type, la variance et la moyenne des résultats de chaque méthode sont résumés en Table 4.5.

| | Méthodes | | | |
|---|---|---|---|---|
| | $E_1$ | $E_2$ | $E_3$ | $E_4$ |
| Écart-type | 2,83 | 2,82 | 2,85 | 2,81 |
| Variance | 8,03 | 7,96 | 8,14 | 7,92 |
| Moyenne | 4,79 | 4,78 | 4,84 | 4,78 |
| $\alpha = 1\%$ | $[3,74;5,84]$ | $[3,74;5,83]$ | $[3,78;5,90]$ | $[3,74;5,83]$ |
| $\alpha = 5\%$ | $[3,99;5,59]$ | $[3,99;5,58]$ | $[4,03;5,65]$ | $[3,99;5,58]$ |

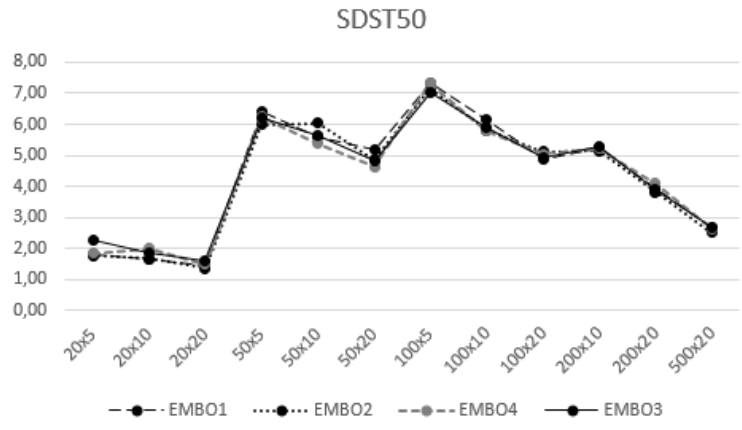
**Tableau 4.5: Indicateurs d'efficacité complémentaire et intervalles de confiance pour les méthodes $EMBO_i$, $i = 1,\ldots,4$, avec $\alpha = 1\%,5\%$.**

La Figure 4.6 illustre les pourcentages de déviation moyens pour chaque taille d'instance et pour chaque groupe d'instances (SDST10, SDST50, SDST100 and SDST125). Dans chaque figure, l'axe $Ox$ représente les différentes tailles d'instances étudiées. Lesdites tailles sont sous le format : nombre de jobs $\times$ nombre de machines. L'axe $Oy$ représente les pourcentages de déviation moyens pour chaque taille d'instance.

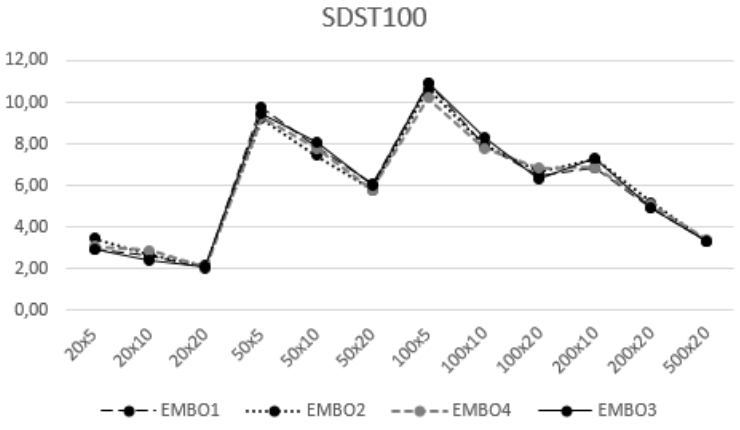
D'après les résultats de la Table 4.4 et de la Figure 4.6, nous pouvons dire que nous avons amélioré la version de base de façon significative. En effet, les méthodes $EMBO_i$, $i = 1,\ldots,4$ surpassent de loin la méthode $BMBO$ ; ils parviennent même à trouver la meilleure solution connue pour deux instances du problème étudié. Dans les autres cas, le pourcentage de déviation moyen est proche de 0% et ne dépasse pas 13% dans les pire cas.



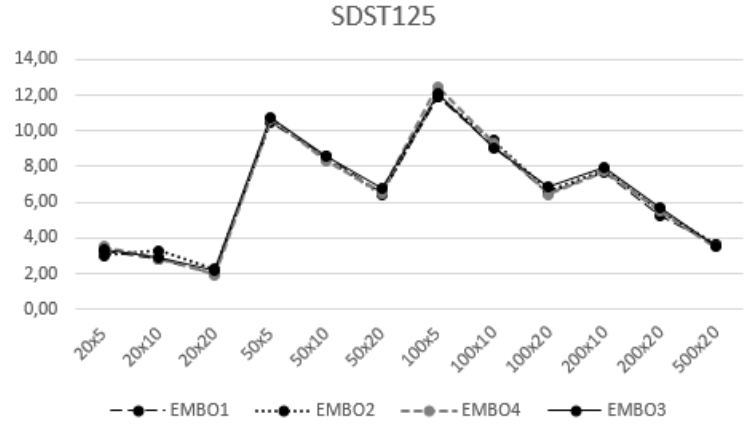

**(a) Résultats moyens sur les instances SDST10.**




**(b) Résultats moyens sur les instances SDST50.**




**(c) Résultats moyens sur les instances SDST100.**




**(d) Résultats moyens sur les instances SDST125.**

**Figure 4.6: Résultats moyens pour les méthodes $EMBO_i$, $i = 1, \ldots, 4$Réglage des paramètres de $BMBO$.**

Dans la Table 4.5, nous présentons les intervalles de confiance pour la moyenne des quatre méthodes avec des seuils de signification de $\alpha = 1\%$ et $\alpha = 5\%$ (niveaux de confiance de 99% et 95%, respectivement).

Nous observons que des résultats de bonne qualité sont produits avec des niveaux de confiance significatifs. En effet, d'après la Table 4.5, nous observons que les méthodes $EMBO_i$, $i = 1,\ldots,4$ produisent des résultats moyens inférieur à 6% avec des niveaux de confiance de 95% et 99%. Nous remarquons aussi que le pourcentage de déviation moyen pour chaque méthode *EMBO* est assez stable ; il se situe entre 3.7% et 5.9% avec un niveau de confiance de 99%.

Les exécutions prennent environs 80 secondes pour les plus petites instances et ne dépassent pas 80 minutes pour les plus grandes.

Ensuite, nous avons voulu comparer les performances individuelles des nos méthodes *EMBO*. Pour ce faire, nous avons fait appel à des tests d'hypothèse. Les méthodes ont été comparées deux à deux. Pour comparer deux méthodes $EMBO_i$ et $EMBO_j$, $1 \leq i, j \leq 4$ et $i \neq j$ ; notre hypothèse nulle est l'égalité des pourcentages de déviation moyens de $EMBO_i$ et de $EMBO_j$. Après avoir effectué le test de Student [Fay et Proschan (2010)], nous avons accepté cette hypothèse pour chaque couple de méthodes *EMBO* avec des seuils de signification de $\alpha = 0,01$ et $\alpha = 0,05$, respectivement. Ainsi, aucune des quatre méthodes ne domine les autres ni est dominées par elles. Nous avons confirmé ce résultat avec un test de Wilcoxon [Golden et Stewart (1985)] au même seuil de signification.

Nous avons donc quatre méthodes qui fournissent des résultat structurellement différents mais de qualité équivalente, ce qui est un résultat intéressant dans le sens où il accorde plus de flexibilité lors de la prise de décision.

Nous avons aussi observé que nos méthodes génèrent de bons résultats pour les plus grandes instances. De plus, leur efficacité augmente proportionnellement au nombre de machines ; ceci en fait de bonnes candidates pour la résolution de systèmes industriels.

D'un côté, nous savons par [Gupta et Darrow (1986)] que, dans divers problèmes réels, les temps de réglage sont relativement petits en comparaison avec les temps de traitement. Alors les instances des types $SDST10$ et $SDST50$ sont plus susceptibles d'être rencontrées en pratique. D'un autre côté, il est intéressant de noter que la moyenne de nos algorithmes *EMBO* n'excèdent 7% pour ces groupes d'instances. Ainsi, il y a de bonnes chances pour que nos méthodes soient efficaces pour la résolution de problèmes réels.

## 4.5 CONCLUSION

Dans ce chapitre, nous avons adapté la méthode MBO pour la résolution du problème de flow shop de permutation avec temps de réglages où nous avons autant d'opérateurs que de machines. Durant le processus de conception, nous avons cherché à utiliser les propriétés structurelles du problème étudié pour obtenir une méthode adaptée à sa résolution.

La méthode MBO de base a donné des résultats de qualité acceptable sur un ensemble réduit d'instances. Nous avons par la suite conçu une version améliorée dans laquelle nous avons utilisé un nouveau voisinage appelé $4Or$-opt et où la solution de tête était générée par une heuristique ad-hoc. Cette seconde version a surpassé de loin la première et a présenté d'intéressantes caractéristiques quant à une éventuelle application pour la résolution de problèmes réels. Une manière directe d'améliorer la qualité des solutions de nos méthodes serait d'augmenter $K$, le nombre d'évaluations de solutions. En effet, nous étions parvenus à améliorer la performance de *BMBO* dans [Benkalai et al. (2016c)] en augmentant $K$ de 6000 à 10000. Quant à la comparaison avec d'autres approches, elle n'a pu être accomplie car les précédentes études

traitant du problème de flow shop de permutation avec des temps de réglages ne fournissaient pas assez de détails pour être reproductibles.

Une autre manière d'obtenir des résultats encore meilleurs serait de construire d'autres structures de voisinage qui seraient plus adaptées à la complexité inhérente du problème. Il serait aussi intéressant d'ajouter d'autres ingrédients au MBO de base, tels que la restructuration de la population après un certain nombre d'itérations sans amélioration. Ceci permettrait de guider la recherche vers une autre zone de l'espace de recherche et ainsi améliorer l'exploration de ce dernier.

Nous pourrions aussi envisager l'hybridation de l'approche MBO avec d'autres métaheuristiques telles de que la recherche à voisinage variable et/ou des méthodes exactes telles que le branch-and-bound. Ceci permettrait d'améliorer simultanément les capacités d'exploitation et d'amélioration de la méthode.

# CHAPITRE 5

# FLOW SHOPS AVEC OPÉRATEURS

## 5.1 INTRODUCTION

Lors de la gestion d'un système de production, on peut avoir besoin, en plus des machines, de ressources additionnelles. Ces ressources peuvent aller d'un opérateur humain à un robot en passant par un outil ou un logiciel informatique [Hall et al. (2000); Blazewicz et al. (2007)]. Cependant, l'humain a toujours joué un rôle crucial en économie, étant à la fois producteur et consommateur mais aussi le seul à pouvoir faire face aux aléas qui peuvent se présenter, et ce, malgré les progrès accomplis en équipements de production [Baptiste et al. (2005)].

D'étroits liens existent entre les ressources humaines et matérielles dans les systèmes industriels, particulièrement au niveau opérationnel ; de nombreux travaux ont ainsi souligné la nécessité d'intégrer les ressources humaines lors de l'ordonnancement de systèmes de production [Zouba (2009)]. Cependant, la considération des ressources humaines et de leurs différentes caractéristiques, telles que le niveau d'expérience, leur aptitudes et la charge de travail variable qui leur est assignée rend les problèmes d'ordonnancement résultats plus complexes.

Néanmoins, les modèles d'ordonnancement "classiques" [1] ont tendance à être irréaliste ; dans l'environnement économique actuel qui est fortement concurrentiel, il devient de plus en plus important de développer des modèles plus efficaces et plus réalistes. Une manière d'y parvenir est d'intégrer les ressources humaines lors de la prise de décisions en ordonnancement de ressources matérielles pour ainsi réduire l'écart entre les modèles étudiés en théorie et les problèmes réels observés en pratique. Dans ce contexte, on aurait alors à concevoir des emplois du temps pour les opérateurs mais aussi à les affecter à des postes de travail.

En ordonnancement classique, il est supposé implicitement que le nombre d'opérateurs est égal au nombre de machines, ce qui est rare dans les problèmes réels. Dans le présent chapitre, nous traitons le cas où le nombre d'opérateurs est inférieur au nombre de machines. Nous y étudions l'impact de la considération simultanée des ressources matérielles et humaines sur la qualité des solutions produites. D'abord, nous comparons des approches qui affectent des opérateurs sur une séquence donnée de jobs générée par des heuristiques pour le problème du flow shop classique. Les résultats obtenus sont comparés à une borne inférieure conçue pour le problème à l'étude. Nous cherchons à évaluer les importances respectives des deux aspects du problème, à savoir, la gestion des ressources humaines et celle des ressources matérielles. Ensuite, nous comparons les approches de résolution simultanée, où les opérateurs sont affectés en même temps que les jobs sont ordonnancés et l'approche séquentielle, où les opérateurs sont affectés étant donnée une séquence de jobs. Nous cherchons à savoir s'il est préférable de considérer le problème comme un tout ou alors de le diviser en phases.

1. Dans le reste du mémoire, nous utilisons le terme "classique" pour les problèmes sans contraintes de ressources.

## 5.2 DESCRIPTION DES PROBLÈMES ET NOTATION

Le problème du flow shop classique peut être formulé comme suit. Étant donnés un ensemble $J = \{1,2,\ldots,n\}$ de $n$ jobs et un ensemble $M = \{M_1,M_2,\ldots,M_m\}$ de $m$ machines. Chaque job, $j$, doit être traité sans interruption sur chaque machine, il est composé de $m$ opérations, $O_{1j},\ldots,O_{mj}$. Pour chaque opération, $O_{ij}$, est associé un nombre $p_{i,j} \in \mathbb{Z}^+$, qui représente le temps de traitement de ladite opération sur la machine $M_i$. Chaque job est d'abord traité sur la machine $M_1$, ensuite sur la machine $M_2$ et ainsi de suite jusqu'à sa complétion sur la machine $M_m$. Nous supposons dans le cadre de nos travaux qu'un job nécessite la présence de l'un des $k$ opérateurs pour toute la durée de son traitement. En d'autres termes, une machine restera inactive tant qu'il n'y a aucun opérateur de disponible pour la faire fonctionner. Ainsi, les temps de traitement ne sont pas directement affectés par l'intervention des opérateurs. L'impact du partage d'opérateurs est modélisé par les temps d'attente engendrés par le fait que le nombre d'opérateurs est inférieur au nombre de machines.

Nous avons étendu la notation présentée dans [Blazewicz et al. (2007)] pour les problèmes d'ordonnancement avec contraintes de ressources afin de pouvoir décrire les caractéristiques de nos problèmes en ajoutant les symboles suivants :

— $S$ pour indiquer que le problème se résout étant donnée au départ une séquence de jobs.
— $e$ pour indiquer que l'on utilise le mode de changement d'affectation en *fin de tâche (end of task)*, où un opérateur ne peut interrompre l'exécution de : l'opération de laquelle il est en charge jusqu'à la complétion de cette dernière. L'opérateur pourra alors commencer le traitement d'une opération sur la même machine, sur une autre machine ou simplement rester inactif.
— $f$ pour indiquer que l'on utilise le mode de changement d'affectation *libre (free)*, où un opérateur peut interrompre l'opération qu'il exécute à n'importe quel moment pour

passer à une autre ou simplement rester inactif. L'opération interrompue pourra par la suite être reprise par le même opérateur ou par un autre.

— $v_h$ pour indiquer, dans le cas où l'on a différents types d'opérateurs, que ces derniers ont différents niveaux de performance (vitesses de travail).

Dans nos travaux, nous avons développé un ensemble de bornes inférieures pour nous permettre d'évaluer la qualité de nos méthodes de résolution, elles sont de la forme $\beta_{ou}(k)$ où :

— $o$ précise les contraintes de l'environnement et est égal à $s$, $a$ ou . respectivement, si le problème est soumis aux contraintes d'une séquence de départ $S$, d'une affectation $A$ ou alors considère les deux aspects simultanément.

— $u$ précise le mode de changement d'affectation et peut être égal à $e$ ou $f$ tel que décrit en Section 5.2.

— $k$ est le nombre d'opérateurs.

## 5.3 MODE DE CHANGEMENT D'AFFECTATION EN FIN DE TÂCHE

Dans cette section, nous considérons que le changement d'affectation des opérateurs sur les différentes machines se fait selon le mode en *fin de tâche i.e.* un opérateur ne peut pas interrompre le traitement d'une opération avant sa complétion. Les résultats ont été publiés dans [Benkalai et al. (2015, 2017b)].

### *5.3.1 MINIMISATION DU MAKESPAN*

Nous notons le problème d'affectation d'opérateurs étant donnée une séquence de jobs par $F_m|res\ 1k1, S, e|C_{max}$. Comme nous traitons le problème du flow shop standard [2] avec opéra-

2. Sans contraintes de permutation.

teurs, alors la séquence donnée est notée $S = (S_{ij})$, $i = 1,\ldots,m$, $j = 1,\ldots,n$.

L'objectif est de trouver une affectation d'opérateurs $a^*$ telle que

$$C_{max}(S,a^*) = \min_{a\in A} C_{max}(S,a),$$

où $A$ représente l'ensemble des affectations possibles et $C_{max}(S,a)$ est le temps de complétion final de la séquence de jobs $S$ sous l'affectation $a$.

Étant donnée une solution, $(S,a)$, les temps de fin de traitement sont calculés comme suit. Avant de poursuivre, nous décrivons d'abord la notation utilisée :

— L'opération $S(i,j)$ correspond à la $i$ème opération du job à la $j$ème position sur la machine $M_i$.

— L'opérateur $h$, $h = 1,\ldots,k$, exécute $n_h$ opérations : $op_c$, $c = 1,\ldots,n_h$.

— $av_{h,op_c,S(i,j)}$ est le temps auquel la $c$ème opération traitée par l'opérateur $h$ [3] devient disponible [4].

— $ts_{h,op_c,S(i,j)}$ est le temps auquel l'opérateur $h$ en charge de l'opération $S(i,j)$ [5] commence le traitement de cette dernière.

— $C(S(i,j),a)$, $1 \le i \le m$, $1 \le j \le n$, sont les temps de complétion des opérations $S(i,j)$ sous l'affectation $a$. Ils sont calculés selon la formule récursive suivante.

3. qui correspond à l'opération $S(i,j)$, $1 \le i \le m$, $1 \le j \le n$.

4. Une opération devient disponible quand toutes les opérations la précédant et toutes les opérations du job $j$ sur les machines $M_{i'}, i' < i$, ont été complétées.

5. qui est sa $c$ème opération.

$$ts_{h,op_0,S(i,j)} = av_{h,op_0,S(i,j)} = -\infty;\ h = 1,\ldots,k, \tag{5.1}$$

$$ts_{h,op_c,S(i,j)} = \max\left\{ts_{h,op_{c-1},S(i',j')} + p_{i',S(i',j')}; 0; av_{h,op_c,S(i,j)}\right\};$$
$$h = 1,\ldots,k;\ c = 1,\ldots,n_h, \tag{5.2}$$

$$av_{h,op_c,S(i,j)} = \max\left\{0; C(S(i-1,j'),a)\right\};\ h = 1,\ldots,k;\ c = 1,\ldots,n_h, \tag{5.3}$$

$$C(S(i,j),a) = ts_{h,op_c,S(i,j)} + p_{i,S(i,j)};\ i = 1,\ldots,m;\ j = 1,\ldots,n, \tag{5.4}$$

où $j'$ est la position du job $S(i,j)$ sur la machine $M_{i-1}$.

Les équations (5.2) calculent les temps auxquels les opérateurs commencent le traitement des opérations leur étant affectées. Pour chaque opérateur, ces temps de début correspondent soit au temps où l'opération en question devient disponible ou au temps où l'opérateur termine l'opération qui la précède ou au temps 0 pour leur première opération.

Les équations (5.3) calculent les dates de disponibilité des opérations. Une opération $S(i,j)$ devient disponible lorsque toutes les opérations la précédant sur la machine $M_i$ ainsi que les opérations du job $j$ sur les machines $M_{i'}$, $i' < i$, ont été complétées.

Finalement, les équations (5.4) calculent les temps de complétion des opérations.

Pour plus de clarté, nous présentons l'exemple suivant avec 3 machines, 3 jobs et 2 opérateurs. Les temps d'exécution sont présentés en Figure 5.1.a. La Figure 5.1.b décrit les contraintes de précédence imposées par la séquence de jobs $S$ ainsi que les "routes" de chaque job à travers le système.

Soit $a$ l'affectation suivante

— L'opérateur 1 traite dans cet ordre les opérations suivantes : $S(1,1)$, $S(1,2)$, $S(3,1)$ et $S(3,2)$.
— L'opérateur 2 traite dans cet ordre les opérations suivantes : $S(2,1)$, $S(2,2)$, $S(1,3)$, $S(2,3)$, et $S(3,3)$.

Le diagramme de Gantt de la solution $(S,a)$ est illustré en Figure 5.1.c. La valeur correspondante du makespan est 9.5.

D'un autre côté, le problème de la gestion simultanée des deux aspects matériel et humain est noté $F_m|res\ 1k1,e|C_{max}$. Cette fois, l'objectif est de trouver une affectation d'opérateurs $a^*$ et une séquence de jobs $S^*$ telles que

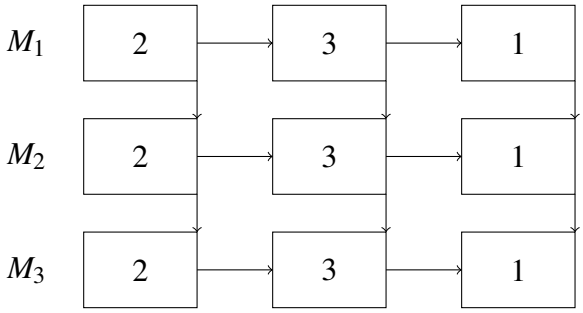
$$C_{max}(S^*,a^*) = \min_{S\in\Pi, a\in A} C_{max}(S,a),$$

où $A$ et $\Pi$ représentent respectivement l'ensemble des affectations possibles et l'ensemble des séquences réalisables, et $C_{max}(S,a)$ est le makespan de la séquence $S$ sous l'affectation $a$.

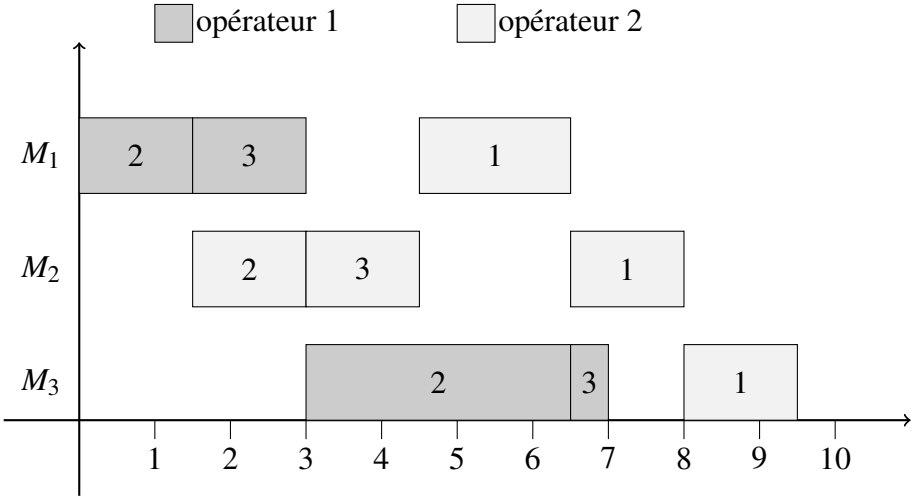
Pour ce problème, les temps de fin de traitement sont calculés comme suit. Nous gardons la même notation décrite précédemment et nous y ajoutons le paramètre $am_i$ qui est le temps le plus récent où une opération a été complétée sur la machine $M_i$, $i = 1,\ldots,m$. Il est mis à jour à chaque fois qu'une opération est complétée sur $M_i$. Les temps de fin de traitement sont calculés selon la formule récursive suivante.

| | | $j=1$ | $j=2$ | $j=3$ |
|---|---|---|---|---|
| $p_{i,j}$ | $i=1$ | 2 | 1.5 | 1.5 |
| | $i=2$ | 1.5 | 1.5 | 1.5 |
| | $i=3$ | 1.5 | 3.5 | 0.5 |

**(a) Données associées avec l'instance $I$.**




**(b) Contraintes de précédence générées par les routes des jobs et par la séquence $S$.**




**(c) Le diagramme de Gantt et le makespan de la solution $(S,a)$.**

**Figure 5.1: Une instance de $F_m|res\ 1k1,S,e|C_{max}$.**

$$ts_{h,op_0,S(i,j)} = av_{h,op_0,S(i,j)} = -\infty;\ h = 1,\ldots,k, \tag{5.5}$$

$$ts_{h,op_c,S(i,j)} = \max\left\{ts_{h,op_{c-1},S(i',j')} + p_{i',S(i',j')}; 0; av_{h,op_c,S(i,j)}\right\};$$
$$h = 1,\ldots,k;\ c = 1,\ldots,n_h, \tag{5.6}$$

$$av_{h,op_c,S(i,j)} = \max\left\{0; C(S(i-1,j'),a); am_i\right\};\ h = 1,\ldots,k;\ c = 1,\ldots,n_h(^*), \tag{5.7}$$

$$C(S(i,j),a) = ts_{h,op_c,S(i,j)} + p_{i,S(i,j)};\ i = 1,\ldots,m;\ j = 1,\ldots,n. \tag{5.8}$$

om $j'$ est la position du job $S(i,j)$ sur la machine $M_{i-1}$.

Les équations (5.6) calculent les temps auxquels les opérateurs commencent le traitement des opérations leur étant affectées. Ce temps correspond soit au moment où l'opération concernée devient disponible ou au moment où l'opérateur termine l'opération qu'il est en train d'exécuter, ou au temps 0 pour leur première opération.

Les équations (5.7) calculent les temps auxquels les opérations sont disponibles. Une opération $S(i,j)$ devient disponible quand la machine $M_i$ le devient et que toutes les opérations du job $j$ sur les machines $M_{i'}$, $i' < i$, ont été complétées.

Enfin, les équations (5.8) calculent les temps de fin de traitement des opérations.

### A. Étude de complexité

Nous commençons l'étude de la complexité avec le cas trivial à un opérateur. Nous montrons que dans ce cas, le problème est résoluble en temps polynomial que la séquence de jobs soit fixée ou pas.

**Cas à un opérateur**

**Lemme 1.** $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$ *est une borne inférieure pour les problèmes* $F_m|res\ 111,S,e|C_{\max}$ *et* $F_m|res\ 111,e|C_{\max}$.

*Démonstration.* : Évidente. □

**Théorème 1.** *Les problèmes* $F_m|res\ 111,S,e|C_{\max}$ *et* $F_m|res\ 111,e|C_{\max}$ *sont résolubles en temps polynomial.*

*Démonstration.* Un algorithme qui fait traiter à l'opérateur, sans temps d'attente, toutes les opérations de la machine $M_1$ puis toutes celles de la machine $M_2$ et ainsi de suite jusqu'à la machine $M_m$ produit une solution avec un makespan égal à la borne inférieure du Lemme 11.Cet algorithme est en $O(mn)$ ; le résultat du théorème suit immédiatement. □

Dans ce qui suit, nous nous concentrons sur la complexité des problèmes avec $k \geq 2$ opérateurs. Nous commençons par le cas où la séquence de jobs est fixée.

**Cas séquentiel avec** $k \geq 2$ Notre preuve consiste à montrer que le problème de décision suivant est $\mathscr{NP}$-complet au sens fort pour un $k \geq 2$ fixé.

$FSMS(k,m)$ est un problème de flow shop avec $\ell$ jobs, $m \geq 3$ machines et $k \geq 2$ opérateurs où l'affectation des opérateurs change selon le mode en fin d'opération et $S$ est une séquence de jobs fixée pour les différentes machines ; étant donné un entier $\alpha$, existe-t-il une affectation $a \in A$ telle que $C_{max}(S,a) \leq \alpha$ ?

Pour ce faire, nous montrons que $FSMS(2,3)$ est $\mathscr{NP}$-complet au sens fort. La réduction est construite à partir d'un problème à deux machines parallèles définit comme suit :

*PM* est un problème à deux machines parallèles avec $n$ jobs et des contraintes de précédence sous forme de $\ell$ chaînes, où $p_h$ est le temps d'exécution du job $h$, $1 \leq h \leq n$. Étant donné un entier $\alpha$, existe-t-il un ordonnancement $\pi$ tel que $C_{max}(\pi) \leq \alpha$ ?

Soit $I$ une instance du problème *PM*. Nous construisons une instance $I'$ de $FSMS(2,3)$ comme suit. Nous notons $C_w$, $w = 1, \ldots, \ell$, les chaînes de $I$. Le système de job de $I'$ contiens $\ell$ jobs où chaque job a une seule opération sur l'une des trois machines :

— si $w \equiv 0 \mod 3$, alors un job $w$ sera traité sur la machine $M_1$ avec un temps de traitement égal à la somme des temps de traitement des jobs de la chaîne $C_w$ ; *i.e.* $p_{1w} = \sum_{h \in C_w} p_h$, $p_{2w} = p_{3w} = 0$.

— si $w \equiv 1 \mod 3$, alors un job job $w$ sera traité sur la machine $M_2$ avec un temps de traitement égal à la somme des temps de traitement des jobs de la chaîne $C_w$, *i.e.* $p_{2w} = \sum_{h \in C_w} p_h$ ; $p_{1w} = p_{3w} = 0$.

— si $w \equiv 2 \mod 3$, alors un job $w$ sera traité sur la machine $M_3$ avec un temps de traitement égal à la somme des temps de traitement des jobs de la chaîne $C_w$, *i.e.* $p_{3w} = \sum_{h \in C_w} p_h$ ; $p_{1w} = p_{2w} = 0$.

De plus, il existe une contrainte de précédence entre les jobs $w$ and $w+3$, $w = 1, \ldots, \ell-3$. Ceci induit la séquence fixée de jobs pour $I'$. Enfin, nous posons $\alpha = \lceil \ell/2 \rceil \max_{1 \leq w \leq \ell} \{\text{Length}(C_w)\}$, où $\text{Length}(C_w)$ est la somme des temps de traitement des jobs dans la chaîne $C_w$. Il est évident que cette construction se fait en temps polynomial. Pour plus de clarté, nous présentons l'exemple suivant de construction.

Soit une instance $I$ de PM avec deux machines, 14 jobs et 6 chaînes. Les temps d'exécution des jobs sont résumés dans la Table 5.1 et les contraintes de précédence dans la Figure 5.2. À partir de $I$, nous construisons une instance $I'$ de $FSMS(2,3)$ avec 6 jobs ; les données de cette instance sont résumées en Table 5.2. Il y a des contraintes de précédence entre les jobs 1 et 4

sur la machine $M_1$, 2 et 5 sur la machine $M_2$ et 3 et 6 sur la machine $M_3$.

| Jobs $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_j$ | 3 | 2 | 1 | 4 | 5 | 1 | 2 | 3 | 2 | 4 | 1 | 1 | 4 | 3 |

**Tableau 5.1: Temps d'exécution des jobs de $I$.**

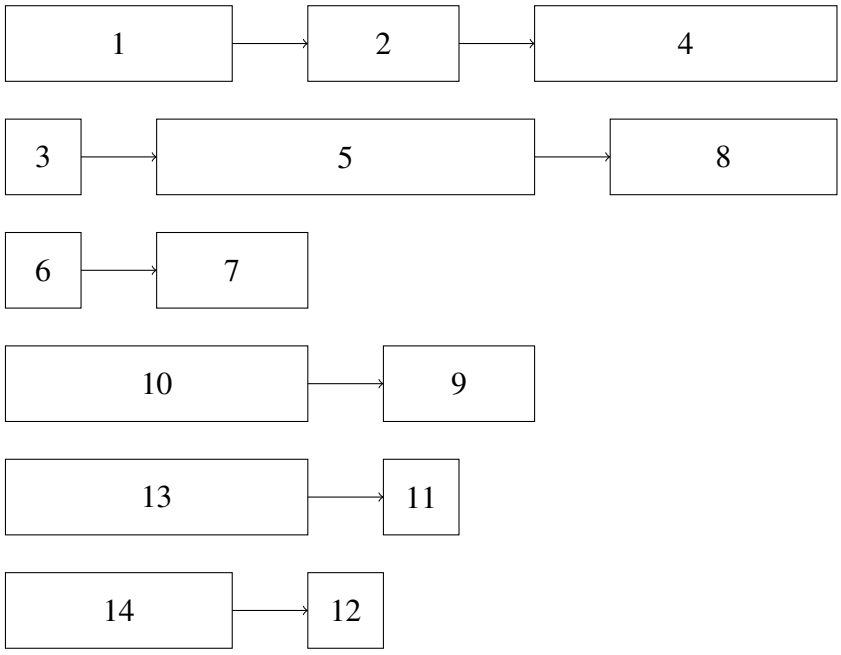



**Figure 5.2: Contraintes de précédence de l'instance $I$.**

| | | Jobs | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 |
| Machines | $M_1$ | 9 | 0 | 0 | 6 | 0 | 0 |
| | $M_2$ | 0 | 9 | 0 | 0 | 5 | 0 |
| | $M_3$ | 0 | 0 | 3 | 0 | 0 | 4 |

**Tableau 5.2: Temps d'exécution des jobs de l'instance $I'$.**

Maintenant, nous montrons que l'instance $I$ a la réponse "oui" si, et seulement si, l'instance $I'$ a la réponse "oui".

**Lemme 2.** *Si I a la réponse "oui", alors il y a au moins une solution où $\lceil \ell/2 \rceil$ chaînes sont exécutées sur la première machine et le reste des chaînes par la seconde machine.*

*Démonstration.* Comme les chaînes sont indépendantes, on peut traiter $\lceil \ell/2 \rceil$ chaînes l'une après l'autre sans dépasser $\alpha$. Le reste des chaînes peut ainsi être exécuté sur la seconde machine toujours sans dépasser $\alpha$. □

**Lemme 3.** *Si I a la réponse "oui", alors $I'$ a la réponse "oui" aussi.*

*Démonstration.* Si $I$ a la réponse "oui", nous avons un ordonnancement $\pi$ sur les deux machines parallèles tel que $C_{max}(\pi) \leq \alpha$. Sans perte de généralité, on suppose que l'ordonnancement $\pi$ a la structure décrite dans le Lemme 12. Cette solution correspond à une solution de $I'$ où l'opérateur 1 (2) traite les jobs de la machine $M_1$ ($M_2$) du problème $PM$. Comme l'ensemble de jobs sur chaque machine du flow shop est indépendant de ceux des autres machines, les opérateurs peuvent alors traiter les chaînes leur étant assignées l'une après l'autre sans problème, ainsi, le temps de traitement total ne dépassera pas $\alpha$. Ainsi, $I'$ a la réponse "oui". □

**Lemme 4.** *Si $I'$ a la réponse "oui", alors il y a au moins une solution telle qu'un opérateur traite $\lceil \ell/2 \rceil$ jobs et l'autre opérateur traite le reste des jobs.*

*Démonstration.* Si l'opérateur 1 traite $\lceil \ell/2 \rceil$ jobs, son temps de travail ne dépasse pas $\alpha$ par définition. Ce sera aussi le cas pour l'opérateur 2 étant donné que ce dernier traite au plus autant de jobs que l'opérateur 1.

□

**Lemme 5.** *Si $I'$ a la réponse "oui", alors I a la réponse "oui".*

*Démonstration.* Si $I'$ a la réponse "oui", nous avons une affectation d'opérateurs $a$ et une séquence de jobs $S$ telles que $C_{max}(S,a) \leq \alpha$. Le traitement des $\ell$ jobs de $I'$ avec deux opérateurs est équivalent au traitement des $\ell$ chaînes de $I$ sur les deux machines du problème

*PM*. Dans *PM*, les jobs de chaque chaîne peuvent ainsi être exécutés sans interruption. Selon le Lemme 15, il s'en suit que dans *PM*, la machine $M_1$ traite $\lceil \ell/2 \rceil$ chaînes, alors que la machine $M_2$ traite le reste des chaînes, produisant ainsi un ordonnancement $\pi$ avec $C_{max}(\pi) \leq \alpha$. Ainsi, *I* a la réponse "oui". □

**Théorème 2.** *Le problème $F_m|res\ 1k1, S, e|C_{max}$ est $\mathscr{NP}$-difficile au sens fort pour $k \geq 2$.*

*Démonstration.* Le problème $FSMS(2,3)$ est clairement dans $\mathscr{NP}$. Par les Lemmes 13 et 15, l'instance *I* de *PM* a une solution si, et seulement si, l'instance $I'$ de $FSMS(2,3)$ a une solution. Le résultat du théorème est donc établi. □

**Cas simultané avec** $k \geq 2$ Dans cette section, nous prouvons que le cas où l'affectation des opérateurs et la construction de la séquence de tâches se font de façon simultanée est $\mathscr{NP}$-difficile. La preuve de $\mathscr{NP}$-difficulté est inspirée de [Agnetis et al. (2011)]. Elle consiste à prouver que le problème de décision suivant est $\mathscr{NP}$-complet pour un $k \geq 2$ fixé :

$FSM(k)$ est un problème de flow shop avec *n* jobs, $m = 3$ machines et $k \geq 2$ opérateurs où le changement d'affectation des opérateurs se fait selon le mode en fin d'opération ; étant donné un entier $\alpha$, existe-t-il une séquence *S* et une affectation $a \in A$ telles que $C_{max}(S,a) \leq \alpha$ ?

Pour la réduction, on utilise le problème de partition défini comme suit :

Étant donné un ensemble fini $A = \{a_1, a_2, \ldots, a_n\}$ de cardinalité *n* et des tailles $s(a_j) \in \mathbb{Z}^+$, $j = 1, 2, \ldots, n$, existe-t-il un sous-ensemble $A' \subseteq A$ tel que $\sum_{a_j \in A'} s(a_j) = \sum_{a_j \in A - A'} s(a_j)$ ?

Soit *I* une instance du problème de partition. Nous construisons une instances $I'$ de $FSM(2)$ comme suit. Le système de jobs contient $n+2$ jobs, chacun de ces jobs ayant une seule opération sur l'une des trois machines.

- $B = \sum_{j=1}^{n} s(a_j)$,
- $p_{1j} = p_{2j} = 0, p_{3j} = s(a_j), j = 1, \ldots, n$,
- $p_{1,n+1} = p_{2,n+2} = L$,
- $p_{2,n+1} = p_{3,n+1} = p_{1,n+2} = p_{3,n+2} = 0$,
- $\alpha = L + (B/2)$, où $L > (B/2), L \in \mathbb{Z}^+$.

Il est évident que cette construction se fait en temps polynomial. Maintenant, nous montrons que l'instance $I$ a la réponse "oui" si, et seulement si, l'instance $I'$ a la réponse "oui".

**Lemme 6.** *Si $I$ a la réponse "oui", alors $I'$ a la réponse "oui".*

*Démonstration.* Si $I$ a la réponse "oui", alors il existe une sous-ensemble $A' \subseteq A$ tel que

$$\sum_{a_j \in A'} s(a_j) = \sum_{a_j \in A - A'} s(a_j) = B/2$$

Ceci veut dire que les jobs de la dernière machine pourraient être divisés en deux sous-ensembles ayant tous deux un temps de traitement total égal à $B/2$ sans causer de préemptions. Alors, l'opérateur 1 peut traiter le job $n+1$ et ensuite les jobs de $M_3$ associés à $A'$, pendant que l'opérateur 2 peut traiter sur $M_3$ les jobs associés à $A - A'$, et ensuite le job $n+2$, tel qu'illustré en Figure 7.1. Cette solution a clairement $C_{max} = L + B/2 \leq \alpha$. Ainsi, $I'$ a la réponse "oui". □

**Lemme 7.** *Si $I'$ a la réponse "oui", alors dans sa solution, chaque opérateur traite une opération de longueur L et des opérations sur la dernière machine de temps de traitement total égal à $B/2$.*

*Démonstration.* Tout d'abord, on montre que chaque opérateur traite au plus une opération de longueur $L$. Si tel n'était pas le cas, le $C_{max}$ résultant sera strictement supérieur à $\alpha$. Ensuite,

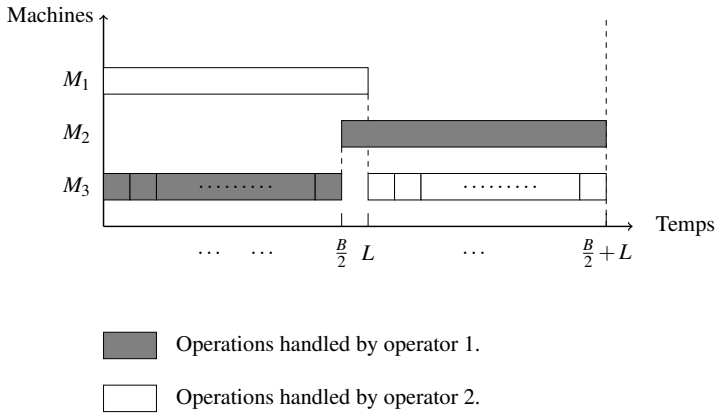



**Figure 5.3: Une solution pour une instance $I'$ du problème $FSM(2)$.**

chaque opérateur traite au moins une opération de taille $L$. Dans le cas contraire, il y aurait au moins une opération de longueur $L$ qui ne serait pas traitée, ce qui donnerait une solution non valide. On en conclut donc que chaque opérateur traite exactement une opération de taille $L$. D'un autre côté, pour une solution avec $C_{max}(S,a) \leq \alpha$, étant donné le résultat ci-dessus, chaque opérateur doit traiter, sur la dernière machine, des opérations dont la somme des temps de traitement est égale à $B/2$. On observe que si une instance $I'$ a une solution, alors le makespan de cette dernière est $C_{max}(S,a) = L + (B/2)$. De plus, aucun des opérateurs ne reste inactif alors qu'une opération est prête pour traitement. Comme les opérations sur la dernière machine ne peuvent pas être traitées simultanément par les deux opérateurs, un opérateur traitera une opération de longueur $L$ alors que l'autre est affecté à la machine $M_3$, ensuite, il changeront de rôle. Une solution $(S,a)$ à l'instance $I'$ serait similaire à celle de la Figure 7.1. □

**Lemme 8.** *Si $I'$ a la réponse "oui", alors $I$ a la réponse "oui".*

*Démonstration.* Si $I'$ a la réponse "oui", alors par le Lemme 17, on sait que, dans une solution de $I'$, chaque opérateur traite exactement une opération de longueur $L$ et des opérations sur $M_3$ de temps de traitement total égal à $B/2$. Comme il n'y a pas de préemption, on en conclut

qu'il existe un sous-ensemble $A' \subseteq A$ tel que $\sum_{a_j \in A'} s(a_j) = \sum_{a_j \in A - A'} s(a_j) = B/2$. Ainsi, $I$ a la réponse "oui". □

**Théorème 3.** *Le problème $F_m|res\ 1k1, e|C_{max}$ est $\mathscr{NP}$-difficile au sens faible pour $k \geq 2$.*

*Démonstration.* Le problème $FSM(2)$ est clairement dans $\mathscr{NP}$. Le résultat du théorème est une conséquence directe des Lemmes 16 et 18.

□

**B. Borne inférieure et heuristiques**

Étant donné la $\mathscr{NP}$-difficulté de nos problèmes, l'utilisation de l'approche heuristique est bien justifiée. Dans cette section, nous présentons quatre heuristiques pour la résolution de nos deux problèmes. Ces heuristiques sont basées sur des règles de priorité gloutonnes. Pour évaluer la qualité de ces heuristiques, nous avons comparé leur performance à une borne inférieure présentée ci-après. Pour plus de clarté, nous allons conclure par un exemple.

**Borne inférieure $\beta_{.e}(k)$** Dans le but d'évaluer la qualité de nos méthodes, nous avons conçu une borne inférieure, notée $\beta_{.e}(k)$, pour $k$ opérateurs. D'abord développée pour le cas à deux opérateurs, cette borne inférieure est basée sur le fait qu'au début et à la fin de l'ordonnancement, seulement un opérateur est actif. Le reste du temps, on voudrait avoir idéalement les deux opérateurs travaillant à plein temps. Cette observation nous donne la borne inférieure suivante.

$$\beta_{\cdot e}(2) = \min_{1\leq j\leq n}\{p_{1,j}\} + \min_{1\leq j\leq n}\{p_{m,j}\} + \frac{\left(\sum_{j=1}^{n}\sum_{i=1}^{m} p_{i,j}\right) - \left(\min_{1\leq j\leq n}\{p_{1,j}\} + \min_{1\leq j\leq n}\{p_{m,j}\}\right)}{2}.$$

En nous basant sur la même remarque, nous avons généralisé cette borne inférieure à $k \geq 2$ opérateurs :

$$\beta_{\cdot e}(k) = \min_{1\leq j\leq n}\{p_{1,j}\} + \min_{1\leq j\leq n}\{p_{m,j}\} + \frac{\left(\sum_{j=1}^{n}\sum_{i=1}^{m} p_{i,j}\right) - \left(\min_{1\leq j\leq n}\{p_{1,j}\} + \min_{1\leq j\leq n}\{p_{m,j}\}\right)}{k}.$$

Dans ce qui suit, nous présentons les heuristiques adaptées pour la résolution de nos problèmes.

**Heuristiques** Quatre heuristiques ont été utilisées pour résoudre le problème d'affectation des opérateurs selon une séquence de jobs donnée. Les trois premières ont aussi été utilisées dans le cas de la gestion simultanée des opérateurs et des jobs.

*Heuristique $H_1$ (Longest Processing Time [Graham (1969)])* : Un opérateur libre choisira l'opération ayant le plus long temps d'exécution.

*Heuristique $H_2$ (Shortest Processing Time [Lee (1991)])* : Un opérateur libre choisira l'opération ayant le plus court temps d'exécution.

*Heuristique $H_3$ :* Un opérateur libre donnera la priorité aux opérations sur la machine ayant le plus petit indice.

*Heuristique $H_4$ :* Dans le cas d'une séquence de jobs fixée, un opérateur libre traitera en priorité les opérations appartenant aux jobs ayant le plus petit indice dans ladite séquence.

Par soucis de clarté, on considère l'exemple suivant avec 4 jobs $(j, j = 1,\ldots,4)$, 4 machines $(M_i, i = 1,\ldots,4)$ et 2 opérateurs. Les temps de traitement sont présentés en Figure 5.4.a.

Soit $S = \{1,2,3,4\}$ la séquence de jobs fixée pour toutes les machines, avec $C_{max}(S)$=18, telle que décrite dans la Figure 5.4.b. Étant donnée $S$, l'affectation des opérateurs avec l'heuristiques $H_2$ génère un $C_{max}(S,a) = 21$ alors que la gestion simultanée de l'ordonnancement et de l'affectation selon l'heuristique $H_1$, produit une solution avec $C_{max}(S,a) = 23$. L'application de ces heuristiques revient à choisir $k$ jobs à exécuter selon la règle de décision de chaque méthode. Les solutions obtenus sont décrites par les Figures 5.4.c et 5.4.d. On remarque que la solution optimale, $(S^*,a^*)$, est produite par l'heuristique $H_2$ étant donnée $S$, en effet, on trouve pour cette instance que $\beta_e(2) = 21$.

### C. Étude expérimentale

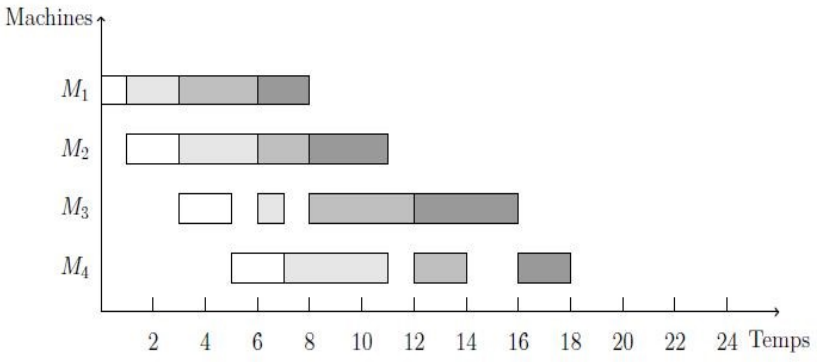
Les heuristiques ont été codées en C++ et déboguées avec Microsoft Visual Studio 2015 sur un PC Lenovo avec un processeur Intel $^{TM}$ Core$^{TM}$ i5-2520M à 2.50 GHz et une RAM de 12 GB. Dans ce qui suit, nous présentons une description de notre benchmark et l'analyse des résultats obtenus.

**Description du benchmark** Comme nous n'avions pas de benchmark de référence, nous avons adapté le benchmark de flow shop de permutation de Taillard, que l'on peut trouver sur le site : `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`.

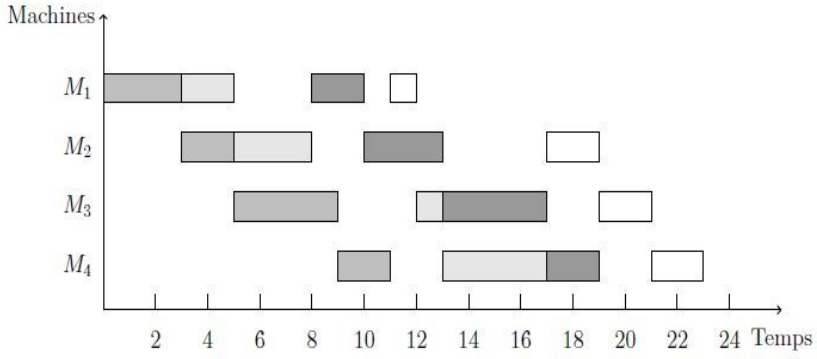
Nous avons choisi un ensemble de 120 instances pour lesquels les combinaisons disponibles pour $n \times m$ sont $\{20,50,100\} \times \{5,10,20\}$, $200 \times \{10,20\}$, et $500 \times 20$.

| | | Jobs | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| Machines | $M_1$ | 1 | 2 | 3 | 2 |
| | $M_2$ | 2 | 3 | 2 | 3 |
| | $M_3$ | 2 | 1 | 4 | 4 |
| | $M_4$ | 2 | 4 | 2 | 2 |

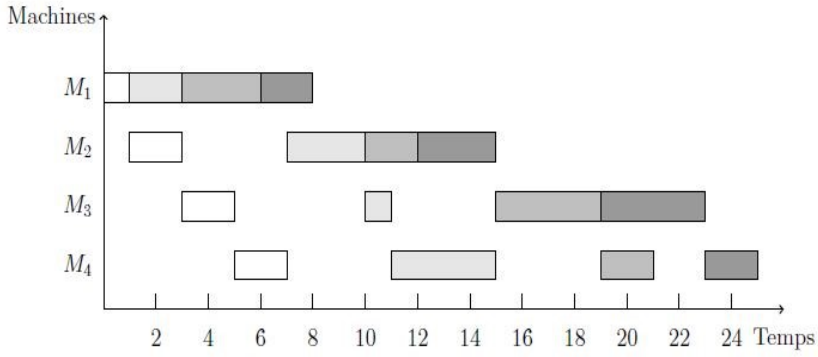
**(a) Temps de traitement de $I$.**




**(b) Diagramme de Gantt pour la solution $S$ sans contraintes d'opérateurs.**




**(c) Diagramme de Gantt pour la gestion simultanée des opérateurs et des jobs avec $H_1$.**




**(d) Diagramme de Gantt pour la solution de $H_2$ étant donnée $S$.**

**Figure 5.4: Exemples de solutions des heuristiques sur l'instance $I$.**

Pour commencer, nous avons utilisé trois heuristiques pour générer les séquences de jobs de départ, à savoir, NEH [Nawaz et al. (1983)], HFC [Koulamas (1998)] et un algorithme de génération aléatoire ; pour les besoins de l'expérimentation, nous avons généré 100 solutions aléatoires par instance.

Rappelons que l'heuristique NEH trie les jobs selon l'ordre décroissant de leur temps de traitement total. Ensuite, suivant cet ordre, elle trouve la meilleure position d'insertion pour chaque job. L'heuristique HFC est basée sur une précédente conjecture de Johnson sur le problème $F_3||C_{max}$ (qui a par la suite été prouvée être fausse par [Burns et Rooker (1976)]). Cette conjecture disait que si la séquence optimale sur les deux premières machines était identique à la séquence optimale sur les deux dernières machines, alors elle était optimale pour le problème global [Johnson (1954)] ; elle a été étendue heuristiquement à des paires de jobs. Finalement, dans l'algorithme de génération aléatoire, la solution est construite en triant les jobs dans un ordre quelconque ; la séquence obtenue sera identique pour chaque machine [6].

Nous avons testé différentes possibilités pour l'affectation des opérateurs. Tout d'abord, l'affectation des opérateurs se faisait sur une séquence fournie par NEH, HFC ou des solutions aléatoires respectivement en utilisant les heuristiques décrites à la Section 5.3.1. Ensuite, nous avons testé la gestion simultanée des opérateurs et des jobs avec les heuristiques $H_1$, $H_2$ et $H_3$.

Les résultats fournis par ces méthodes, présentés en Table 5.5 et 5.6, sont des pourcentages de déviation moyens relativement à la borne inférieure $\beta_e(k)$ présentée à la Section 5.3.1. Ils sont calculés selon la formule suivante

$$d = \frac{(C_{max}^{H_i}(S) - \beta_e(k)) * 100}{\beta_e(k)},\ i \in \{1,2,3,4\},\ k \geq 2,$$

6. Ces solutions respectent la contrainte de permutation. En effet, nous avions remarqué que la génération de solution aléatoire pour le flow shop standard fournissait des résultats de mauvaise qualité.

où $C_{max}^{H_i}(S)$ est le makespan de la solution produite par l'heuristique $H_i$, $i = 1, \ldots, 4$.

Les instances sont notées $n \times m$, où $n$ et $m$ représentent respectivement le nombre de jobs et le nombre de machines du problème de flow shop. "Alea" représente les solutions aléatoires alors que "Simul" indique les solutions des heuristiques gérant simultanément les opérateurs et les jobs. Ainsi, nous avons quatre catégories de méthodes, à savoir, $NEH - H_i$, $HFC - H_i$, $Alea - H_i$, $i = 1, \ldots 4$ et *Simul*. Dans les Figures 5.5, 5.6, 5.7, 5.8, 5.9 et 5.10, l'axe des $y$ représente les pourcentages de déviation moyens et l'axe des $x$ représente le nombre d'opérateurs pour les différentes instances. Pour chaque instance, nous avons testé nos méthodes pour un nombre d'opérateurs allant de 2 à $m - 1$.

**Discussion** Nous avons remarqué que $H_4$ génère généralement les meilleurs résultats que la séquence de départ soit fournie par NEH, HFC ou l'algorithme de génération aléatoire (voir Figures 5.5, 5.6 et 5.7). Sa supériorité augmente avec l'augmentation du nombre d'opérateurs. En effet, cette stratégie permet d'activer plus de machines au début de l'ordonnancement ce qui offre plus de possibilités par la suite, particulièrement quand le nombre d'opérateurs est élevé. En d'autres termes, l'heuristique a fourni des solutions avec une utilisation plutôt équilibrée des machines, ce qui correspond à la minimisation du makespan en pratique.

Un autre fait qu'il est important de souligner est que les résultats obtenus sont très proches de la borne inférieure quand le nombre d'opérateurs est réduit. Nous pensons que cela indique que le problème serait plus "facile" dans ces cas-là.

Nous notons aussi que la différence de performance entre les heuristiques séquentielles quand elles utilisent NEH, HFC ou une solution aléatoire diminue avec la diminution du nombre d'opérateurs. Cette remarque combinée à la précédente implique que la partie la plus importante du problème est celle de l'affectation des opérateurs. C'est sur ce domaine que

nous devrions nous concentrer et développer des méthodes de résolution plus efficaces.

En comparant les différentes approches simultanées (voir Figure 5.8), nous remarquons que c'est l'heuristique $H_2$ qui fournit les meilleurs résultats. En effet, le fait d'accorder la priorité aux opérations les plus courtes fait que plus d'opérateur sont actifs plus vite, ce qui réduit le makespan.

Finalement, si nous comparons les meilleures solutions de chaque catégorie (Figure 5.9), nous remarquons que la performance des méthodes augmente avec l'augmentation de la taille des instances. L'heuristique simultanée $H_2$ a tendance à être meilleure pour un nombre réduit d'opérateurs alors que $NEH - H_4$ est meilleur quand le nombre d'opérateurs est plus élevé (voir Figure 5.10).

En effet, si le nombre d'opérateurs est réduit, ils seront faciles à gérer dans l'atelier et ont moins tendance à rester inactifs, ce qui fait que la partie ordonnancement devient moins compliquée. D'un autre côté, quand le nombre d'opérateurs est élevé, il est préférable d'organiser les jobs en utilisant une bonne heuristique avant de commencer l'affectation des opérateurs.

Nos méthodes fournissent en général de bons résultats proches des bornes théoriques, particulièrement pour les instances de grande taille. En effet, elle ne dépassent pas 51%, 24%, 15%, 9% quand $n = 50$, 100, 200 et 500 respectivement. Le seul cas où le pourcentage de déviation fut plus élevé est celui où $n = 20$ et $m = 20$. Nous pensons que ceci est dû au fait que la borne inférieure soit trop petite pour ce groupe d'instances. Cela ne veut pas dire que nos méthodes sont de mauvaise qualité. Ce genre de problème, que nous pouvons appeler "problèmes carrés" ($n = m$), requerrait des études complémentaires dans le but de développer de meilleures bornes inférieures.

Nous présentons en Table 5.3 des indicateurs d'efficacité complémentaires ainsi que des

intervalles de confiance de seuils de signification $\alpha = 1\%$ et $5\%$. Les résultats montrent que la robustesse de nos méthodes augmente proportionnellement à la taille des instances. En effet, nous remarquons une réduction de l'écart type, la moyenne et la qualité des intervalles de confiance. Quand nous considérons les résultats moyens, nous remarquons que $NEH - H_4$ surpasse les autres méthodes, elle permet d'atteindre des résultats moyens inférieurs à $2\%$ avec un seuil de signification $\alpha = 1\%, 5\%$. De plus, parmi les meilleures méthodes de chaque catégorie, les pires résultats ne dépassent pas $70\%$, et cela n'arrive que dans le cas où $n = 20$. Par ailleurs, nous remarquons par la Table 5.4 que nos méthodes sont rapides ; leurs temps d'exécution moyen ne dépasse pas 5 secondes.

Pour finir, mentionnons que le choix du nombre adéquat d'opérateurs reste une décision à prendre en tenant compte de la politique interne de l'entreprise en termes de coûts. En effet, il pourrait être intéressant d'avoir un nombre plus élevé d'opérateurs dépendamment des objectifs et du contexte. Les heuristiques séquentielles (qui sont les meilleures pour ce genre de cas) présentent aussi l'intérêt d'être plus rapide (voir Table 5.4).

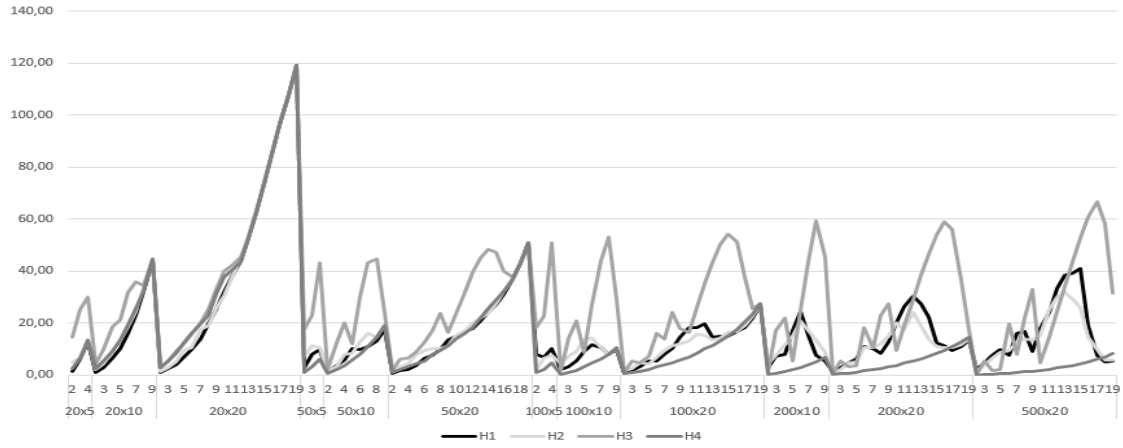



**Figure 5.5: Pourcentages de déviation moyens pour les solutions séquentielles de NEH.**

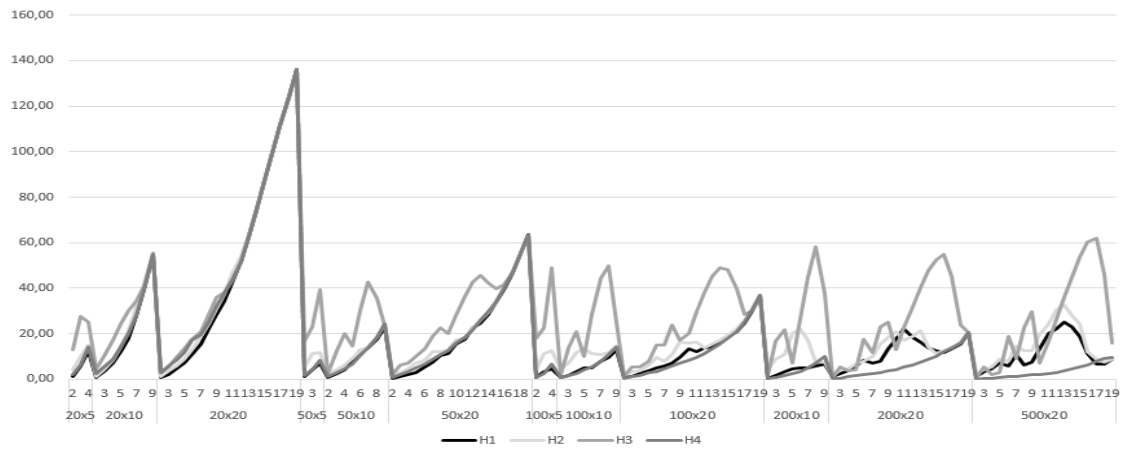



**Figure 5.6: Pourcentages de déviation moyens pour les solutions séquentielles de HFC.**

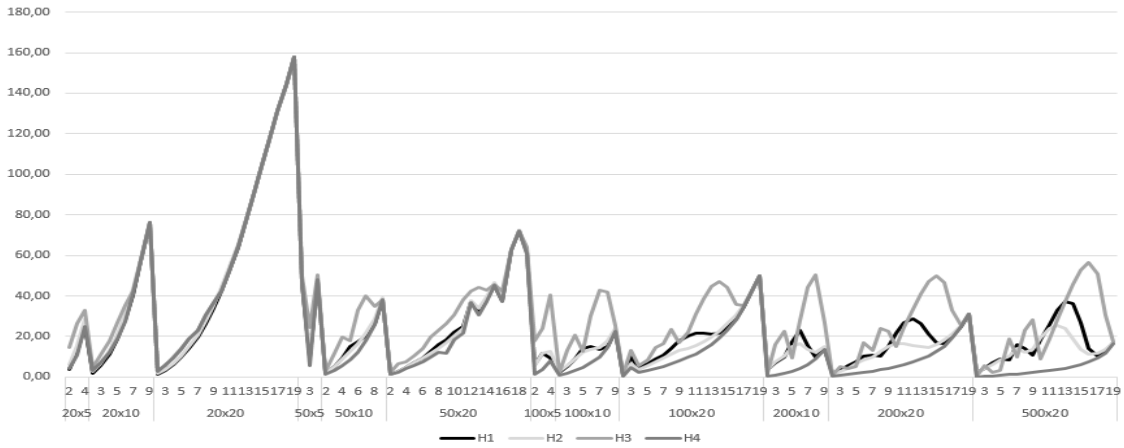



**Figure 5.7: Pourcentages de déviation moyens pour les solutions séquentielles aléatoires.**

## D. Conclusion

Dans cette section, nous avons étudié l'ordonnancement de jobs dans un environnement flow shop avec des opérateurs dont le nombre est inférieur à celui des machines avec pour objectif, la minimisation du makespan. Après avoir prouvé la $\mathscr{NP}$-difficulté de nos problèmes, nous avons comparés deux types d'approches, à savoir, l'affectation d'opérateurs étant donnée une séquence ce jobs et la gestion simultanée des aspects humain et matériel du problème. Les

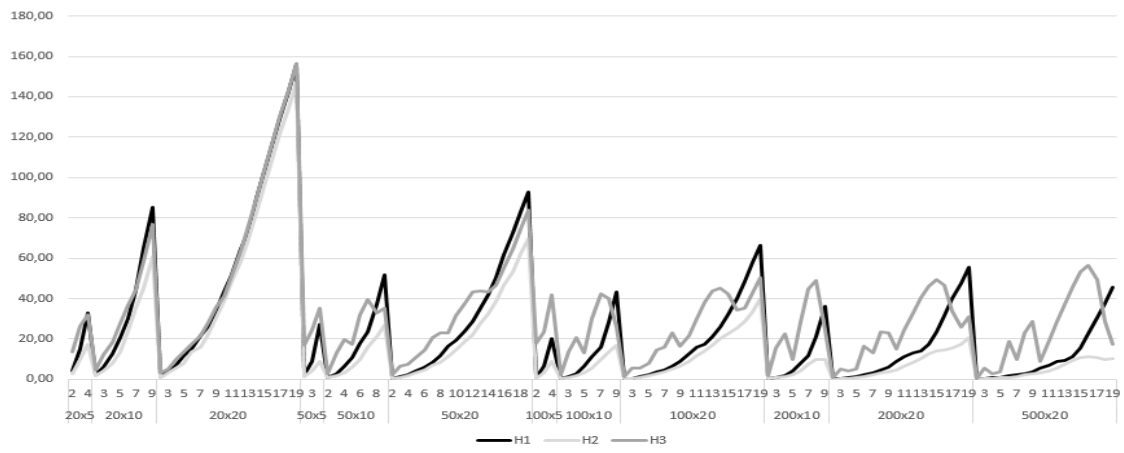



**Figure 5.8: Pourcentages de déviations moyens pour les heuristiques simultanées.**

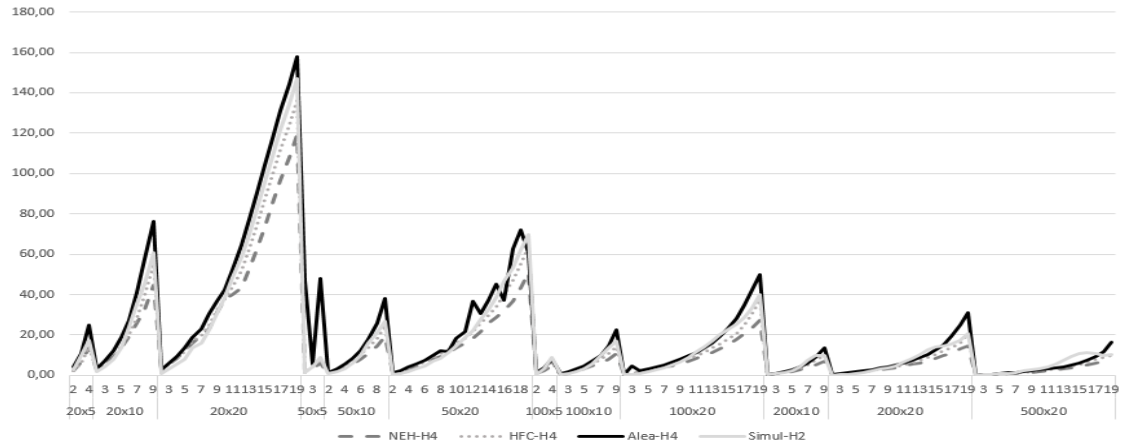



**Figure 5.9: Heuristiques simultanées vs. séquentielles.**

meilleures heuristiques ont fourni de bons résultats proches des bornes inférieures théoriques ; leurs performances étaient également proportionnelles à la taille des instances. Par ailleurs, nous avons trouvé que sur nos méthodes, la meilleure méthode simultanée était meilleure pour les cas à nombre d'opérateurs réduit alors que la meilleure méthode séquentielle était meilleure quand ce nombre était plus grand.

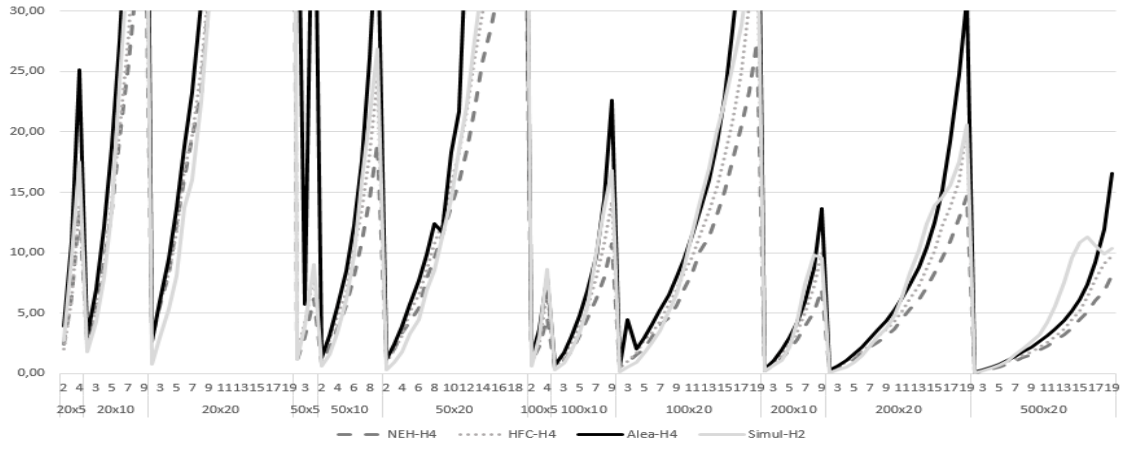



**Figure 5.10: Heuristiques simultanées vs. séquentielles (zoom).**

## 5.4 MODE DE CHANGEMENT D'AFFECTATION LIBRE

Dans la présente section, nous étudions le problème d'ordonnancement de flow shop sous le mode de changement d'affectation libre avec la séquence de tâches donné au départ. Nous considérons tour à tour deux objectifs, la minimisation du makespan puis celle du retard algébrique maximum.

### *5.4.1 MINIMISATION DU MAKESPAN DANS LE CAS SÉQUENTIEL*

L'objet de cette section est l'étude de flow shops avec des opérateurs dont le nombre est inférieur à celui des machines. Nous supposons que le changement d'affectation se fait selon le mode libre. En d'autres termes, un opérateur peut changer d'affectation à n'importe quel moment. Nous supposons également que les temps de traitement sont connus à l'avance. Les résultats ont été présentés dans [Benkalai et al. (2016b)].

Ici encore, nous supposons qu'un job requiert la présence de l'un de $k$ opérateurs durant toute la durée de son traitement. En d'autres termes, une machine restera inactive tant qu'il n'y

| | Méthodes | | | |
|---|---|---|---|---|
| | $n = 20$ | | | |
| | $NEH - H_4$ | $HFC - H_4$ | $Alea - H_4$ | $Simul - H_2$ |
| Écart type | 32,73 | 38,47 | 44,96 | 42,55 |
| Moyenne | 34,96 | 39,52 | 48,47 | 42,29 |
| $\alpha = 1\%$ | $[19,31;50,61]$ | $[21,13;57,92]$ | $[26,97;69,97]$ | $[21,94;62,63]$ |
| $\alpha = 5\%$ | $[23,04;46,87]$ | $[25,52;53,53]$ | $[32,10;64,83]$ | $[26,80;57,77]$ |
| | $n = 50$ | | | |
| Écart type | 13,22 | 16,58 | 20,40 | 19,10 |
| Moyenne | 14,06 | 17,14 | 23,82 | 18,00 |
| $\alpha = 1\%$ | $[7,74;20,38]$ | $[9;21;25,07]$ | $[14,06;33,58]$ | $[8,87;27,13]$ |
| $\alpha = 5\%$ | $[9,25;18,87]$ | $[11,10;23,17]$ | $[16,39;31,25]$ | $[11,05;24,95]$ |
| | $n = 100$ | | | |
| Écart type | 7,13 | 9,17 | 12,58 | 10,74 |
| Moyenne | 7,58 | 9,41 | 12,38 | 10,62 |
| $\alpha = 1\%$ | $[4,18;10,99]$ | $[5,03;13,80]$ | $[6,36;18,39]$ | $[5,49;15,75]$ |
| $\alpha = 5\%$ | $[4,99;10,18]$ | $[6,08;12,75]$ | $[7,80;16,96]$ | $[6,71;14,53]$ |
| | $n = 200$ | | | |
| Écart type | 3,94 | 5,22 | 7,68 | 5,99 |
| Moyenne | 4,56 | 5,73 | 7,54 | 6,61 |
| $\alpha = 1\%$ | $[2,57;6,55]$ | $[3,10;8,37]$ | $[3,67;11,42]$ | $[3,59;9,64]$ |
| $\alpha = 5\%$ | $[3,05;6,07]$ | $[3,73;7,74]$ | $[4,59;10,50]$ | $[4,31;8,92]$ |
| | $n = 500$ | | | |
| Écart type | 2,39 | 2,99 | 4,34 | 4,20 |
| Moyenne | 2,79 | 3,38 | 4,36 | 5,10 |
| $\alpha = 1\%$ | $[1,34;4,24]$ | $[1,56;5,20]$ | $[1,72;6,99]$ | $[2,55;7,65]$ |
| $\alpha = 5\%$ | $[1,69;3,89]$ | $[2,00;4,76]$ | $[2,35;6,36]$ | $[3,16;7,04]$ |

**Tableau 5.3: Indicateurs d'efficacité complémentaires et intervalles de confiance pour la meilleure méthode de chaque catégorie, avec $\alpha = 1\%, 5\%$.**

| | Méthodes | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NEH | | | | HFC | | | | Alea | | | | Simul | | |
| | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ |
| Temps | 1,14 | 1,08 | 0,64 | 0,68 | 1,08 | 1,06 | 0,50 | 0,81 | 1,10 | 1,11 | 0,69 | 0,89 | 4,36 | 4,12 | 1,48 |

**Tableau 5.4: Temps d'exécution moyens.**

a pas d'opérateur pour la faire fonctionner. Ici encore, les temps de traitement ne sont pas directement affectés par l'intervention des opérateurs. L'impact du partage d'opérateurs est modélisé par les temps d'attente occasionnés par le fait qu'il y a moins d'opérateurs que de

| | | Méthodes | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | NEH | | | | HFC | | | | Alea | | | | Simul | | |
| | $k$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ |
| 20x5 | 2 | 1,57 | 4,94 | 14,56 | 2,41 | 1,37 | 4,18 | 12,82 | 2,06 | 3,89 | 6,11 | 14,44 | 3,93 | 4,20 | 2,70 | 13,77 |
| | 3 | 6,01 | 6,40 | 25,31 | 6,38 | 5,96 | 10,05 | 27,66 | 6,17 | 12,16 | 15,45 | 26,49 | 10,88 | 14,88 | 9,64 | 26,38 |
| | 4 | 12,77 | 13,45 | 29,91 | 13,63 | 12,48 | 14,99 | 25,13 | 13,99 | 25,25 | 28,62 | 32,78 | 25,14 | 32,93 | 17,50 | 31,93 |
| 20x10 | 2 | 1,17 | 2,03 | 4,20 | 2,63 | 0,73 | 1,22 | 4,37 | 2,40 | 1,95 | 2,52 | 4,78 | 3,04 | 2,39 | 1,77 | 4,63 |
| | 3 | 2,99 | 4,33 | 10,00 | 5,42 | 3,07 | 4,18 | 10,02 | 5,22 | 5,72 | 6,72 | 10,90 | 6,96 | 6,25 | 4,31 | 12,28 |
| | 4 | 6,37 | 9,89 | 18,60 | 9,22 | 6,87 | 8,33 | 16,99 | 8,52 | 11,44 | 12,96 | 17,96 | 12,12 | 12,67 | 7,91 | 18,13 |
| | 5 | 10,28 | 13,35 | 21,12 | 13,47 | 12,15 | 13,98 | 24,09 | 14,34 | 18,84 | 20,95 | 26,98 | 18,78 | 20,68 | 13,48 | 27,54 |
| | 6 | 16,15 | 18,73 | 31,79 | 19,47 | 18,14 | 21,32 | 30,23 | 20,13 | 28,11 | 31,04 | 35,13 | 27,46 | 30,25 | 24,06 | 36,47 |
| | 7 | 23,06 | 24,43 | 35,93 | 25,69 | 27,91 | 31,58 | 34,13 | 27,98 | 40,68 | 43,36 | 42,33 | 40,84 | 44,01 | 35,05 | 44,09 |
| | 8 | 32,19 | 34,21 | 34,29 | 32,36 | 39,74 | 40,62 | 40,93 | 39,34 | 57,29 | 58,38 | 57,64 | 57,64 | 65,15 | 45,35 | 58,45 |
| | 9 | 44,19 | 44,55 | 44,48 | 44,82 | 55,02 | 55,58 | 55,46 | 54,98 | 76,06 | 76,23 | 76,13 | 76,13 | 85,29 | 60,58 | 76,67 |
| 20x20 | 2 | 0,89 | 1,53 | 2,77 | 2,87 | 0,78 | 1,28 | 2,95 | 2,76 | 1,22 | 1,64 | 3,03 | 2,67 | 1,26 | 0,78 | 3,06 |
| | 3 | 2,47 | 2,74 | 5,25 | 5,53 | 2,05 | 3,80 | 5,42 | 5,78 | 3,36 | 3,82 | 5,50 | 5,94 | 3,88 | 2,99 | 5,22 |
| | 4 | 4,02 | 4,93 | 9,03 | 8,24 | 4,72 | 5,64 | 9,55 | 8,34 | 5,99 | 6,69 | 9,96 | 9,50 | 7,14 | 5,36 | 9,96 |
| | 5 | 6,84 | 9,51 | 12,06 | 12,32 | 7,24 | 8,74 | 13,17 | 11,73 | 9,61 | 10,51 | 13,88 | 13,66 | 11,83 | 8,11 | 13,38 |
| | 6 | 9,84 | 10,53 | 15,64 | 16,17 | 10,99 | 12,90 | 17,48 | 16,92 | 14,29 | 15,05 | 18,68 | 18,77 | 14,84 | 13,74 | 17,56 |
| | 7 | 13,99 | 17,41 | 20,11 | 19,56 | 15,46 | 18,79 | 20,95 | 19,62 | 19,89 | 20,87 | 23,07 | 23,22 | 21,79 | 16,03 | 21,70 |
| | 8 | 19,87 | 19,00 | 25,24 | 23,19 | 22,19 | 24,20 | 28,60 | 25,13 | 26,46 | 27,71 | 30,48 | 30,75 | 26,28 | 22,49 | 28,02 |
| | 9 | 25,16 | 25,52 | 32,90 | 31,27 | 28,49 | 30,77 | 35,85 | 32,32 | 33,75 | 35,27 | 36,36 | 36,49 | 33,88 | 30,49 | 35,57 |
| | 10 | 31,70 | 30,55 | 39,77 | 37,77 | 34,43 | 37,12 | 38,26 | 37,48 | 42,20 | 44,15 | 42,31 | 42,23 | 43,35 | 38,65 | 40,76 |
| | 11 | 37,68 | 37,43 | 42,15 | 40,21 | 42,19 | 46,33 | 43,20 | 43,72 | 52,25 | 54,14 | 52,31 | 52,32 | 52,02 | 48,26 | 51,36 |
| | 12 | 43,84 | 44,75 | 45,28 | 43,45 | 51,70 | 53,80 | 51,86 | 51,69 | 64,13 | 65,29 | 64,40 | 64,42 | 64,33 | 58,17 | 63,58 |
| | 13 | 53,11 | 53,91 | 52,80 | 52,68 | 62,73 | 63,44 | 62,99 | 62,72 | 77,16 | 77,58 | 77,28 | 77,31 | 73,30 | 69,50 | 76,30 |
| | 14 | 62,48 | 62,96 | 62,93 | 62,79 | 74,65 | 74,77 | 74,80 | 74,62 | 90,57 | 90,65 | 90,60 | 90,61 | 89,37 | 82,58 | 89,66 |
| | 15 | 73,69 | 73,87 | 73,65 | 73,58 | 86,98 | 86,98 | 86,98 | 86,98 | 104,04 | 104,05 | 104,05 | 104,04 | 102,61 | 95,56 | 103,08 |
| | 16 | 85,00 | 85,10 | 85,01 | 85,04 | 99,33 | 99,33 | 99,33 | 99,33 | 117,52 | 117,52 | 117,52 | 117,52 | 116,00 | 108,49 | 116,50 |
| | 17 | 96,43 | 96,43 | 96,43 | 96,43 | 111,65 | 111,65 | 111,65 | 111,65 | 130,96 | 130,96 | 130,96 | 130,96 | 129,34 | 121,37 | 129,87 |
| | 18 | 107,85 | 107,85 | 107,85 | 107,85 | 123,96 | 123,96 | 123,96 | 123,96 | 144,40 | 144,40 | 144,40 | 144,40 | 142,68 | 134,25 | 143,24 |
| | 19 | 119,28 | 119,28 | 119,28 | 119,28 | 136,27 | 136,27 | 136,27 | 136,27 | 157,83 | 157,83 | 157,83 | 157,83 | 156,02 | 147,12 | 156,60 |
| 50x5 | 2 | 3,01 | 7,14 | 17,36 | 1,11 | 1,04 | 5,27 | 16,66 | 1,76 | 46,30 | 47,42 | 49,68 | 47,16 | 2,20 | 1,14 | 17,00 |
| | 3 | 7,91 | 11,41 | 22,96 | 3,08 | 4,29 | 11,28 | 23,07 | 4,26 | 9,89 | 11,52 | 24,59 | 5,69 | 8,71 | 3,93 | 24,51 |
| | 4 | 9,65 | 10,66 | 43,33 | 6,16 | 6,93 | 11,74 | 39,13 | 8,14 | 47,00 | 48,16 | 50,42 | 47,95 | 27,21 | 9,05 | 35,15 |
| 50x10 | 2 | 1,36 | 1,61 | 2,20 | 0,79 | 0,77 | 2,29 | 2,34 | 1,28 | 2,11 | 2,25 | 2,78 | 1,29 | 1,03 | 0,64 | 2,79 |
| | 3 | 3,67 | 3,18 | 12,46 | 2,22 | 2,48 | 4,30 | 12,54 | 2,85 | 5,27 | 4,90 | 11,85 | 3,17 | 2,72 | 1,56 | 12,90 |
| | 4 | 5,89 | 8,07 | 20,28 | 3,64 | 4,28 | 6,30 | 19,89 | 4,71 | 9,58 | 8,78 | 19,56 | 5,52 | 6,73 | 3,33 | 19,72 |
| | 5 | 10,13 | 8,41 | 12,03 | 5,68 | 7,87 | 9,09 | 14,70 | 6,78 | 14,94 | 12,46 | 17,80 | 8,48 | 10,98 | 6,10 | 17,22 |
| | 6 | 9,86 | 12,72 | 29,96 | 8,08 | 12,32 | 12,36 | 30,29 | 10,30 | 17,54 | 16,60 | 32,74 | 12,29 | 17,67 | 9,83 | 32,06 |
| | 7 | 10,98 | 15,89 | 43,30 | 10,88 | 13,63 | 14,34 | 42,83 | 13,57 | 19,98 | 21,42 | 40,10 | 17,62 | 23,67 | 16,00 | 39,42 |
| | 8 | 13,01 | 14,65 | 44,52 | 14,45 | 17,55 | 18,30 | 35,63 | 18,16 | 26,37 | 28,68 | 34,79 | 26,06 | 37,31 | 21,22 | 33,04 |
| | 9 | 17,62 | 19,26 | 24,03 | 19,04 | 22,94 | 24,83 | 23,49 | 24,36 | 37,93 | 38,85 | 38,15 | 38,11 | 51,68 | 26,84 | 35,48 |
| 50x20 | 2 | 0,59 | 1,33 | 1,94 | 1,09 | 0,50 | 2,14 | 2,05 | 0,97 | 1,27 | 1,43 | 2,22 | 1,13 | 0,40 | 0,29 | 2,28 |
| | 3 | 1,77 | 3,34 | 6,17 | 2,00 | 1,06 | 2,88 | 6,05 | 1,93 | 2,67 | 2,92 | 6,32 | 2,44 | 1,40 | 0,93 | 6,31 |
| | 4 | 2,31 | 4,27 | 6,65 | 3,33 | 1,88 | 4,51 | 7,21 | 3,13 | 4,44 | 4,85 | 7,42 | 3,95 | 2,24 | 1,76 | 7,51 |
| | 5 | 3,77 | 7,77 | 8,77 | 4,32 | 3,04 | 7,11 | 10,01 | 5,10 | 6,77 | 6,95 | 10,52 | 5,69 | 4,05 | 3,28 | 10,81 |
| | 6 | 6,46 | 9,63 | 12,78 | 5,41 | 5,48 | 8,50 | 13,47 | 6,49 | 9,48 | 9,20 | 14,09 | 7,65 | 6,07 | 4,50 | 14,48 |
| | 7 | 7,70 | 10,33 | 17,26 | 7,86 | 7,65 | 12,08 | 18,69 | 8,55 | 12,48 | 11,60 | 19,64 | 9,84 | 8,63 | 6,89 | 20,55 |
| | 8 | 10,00 | 10,13 | 23,89 | 9,57 | 10,42 | 12,24 | 22,48 | 10,74 | 15,81 | 14,12 | 23,21 | 12,38 | 11,82 | 8,52 | 23,23 |
| | 9 | 13,38 | 11,33 | 16,31 | 11,44 | 11,37 | 12,94 | 20,16 | 12,42 | 18,27 | 16,78 | 26,45 | 11,71 | 16,23 | 11,34 | 22,85 |
| | 10 | 15,07 | 15,19 | 23,33 | 13,80 | 15,60 | 17,02 | 28,19 | 15,77 | 21,87 | 20,08 | 30,76 | 18,27 | 19,29 | 14,43 | 31,61 |
| | 11 | 16,83 | 16,71 | 31,88 | 15,89 | 17,49 | 18,55 | 36,23 | 18,32 | 24,86 | 24,05 | 37,95 | 21,72 | 24,15 | 18,54 | 37,86 |
| | 12 | 17,73 | 20,21 | 39,53 | 18,43 | 22,52 | 22,70 | 42,71 | 21,72 | 37,20 | 37,79 | 42,15 | 36,77 | 28,54 | 22,26 | 43,32 |
| | 13 | 20,69 | 21,88 | 45,22 | 21,82 | 24,84 | 26,04 | 45,58 | 26,08 | 32,23 | 33,73 | 44,27 | 30,42 | 35,51 | 27,55 | 43,97 |
| | 14 | 24,22 | 23,82 | 48,25 | 25,74 | 28,57 | 29,43 | 42,39 | 29,79 | 37,92 | 39,64 | 42,74 | 36,93 | 42,04 | 32,35 | 43,07 |
| | 15 | 27,01 | 27,52 | 47,34 | 28,73 | 33,93 | 33,95 | 39,69 | 34,01 | 45,06 | 46,53 | 45,53 | 45,02 | 50,21 | 38,40 | 46,46 |
| | 16 | 31,17 | 32,26 | 39,95 | 32,09 | 39,24 | 40,82 | 41,45 | 39,91 | 37,81 | 38,39 | 42,49 | 37,40 | 61,76 | 46,37 | 55,30 |
| | 17 | 36,87 | 37,13 | 37,67 | 36,82 | 46,79 | 47,29 | 46,83 | 47,15 | 62,79 | 63,01 | 62,82 | 62,86 | 72,37 | 53,20 | 64,70 |
| | 18 | 43,21 | 43,64 | 43,36 | 43,26 | 55,08 | 55,18 | 55,09 | 55,16 | 72,27 | 72,31 | 72,27 | 72,28 | 82,79 | 62,08 | 74,28 |
| | 19 | 50,98 | 50,94 | 50,96 | 50,93 | 63,63 | 63,63 | 63,63 | 63,63 | 61,60 | 61,62 | 64,10 | 60,95 | 92,93 | 69,69 | 83,94 |

**Tableau 5.5: Pourcentages de déviation moyens du makespan des solutions comparé à la borne inférieure $\beta_e(k)$.**

machines.

Les paramètres $p_{ij}$ et $C_{ij}$ représentent respectivement le temps de traitement et le temps de complétion de l'opération $(i, j)$.

Nous supposons que nous avons $\ell$ types d'opérateurs, $k_h$, $h = 1, \ldots, \ell$ opérateurs de chaque

| | | Méthodes | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | NEH | | | | HFC | | | | Alea | | | | Simul | | |
| | $k$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_1$ | $H_2$ | $H_3$ |
| 100x5 | 2 | 7,92 | 2,05 | 18,36 | 0,90 | 0,84 | 4,25 | 18,09 | 0,85 | 6,26 | 5,46 | 17,67 | 1,36 | 1,02 | 0,61 | 18,04 |
| | 3 | 6,96 | 6,58 | 22,52 | 2,29 | 3,13 | 11,42 | 22,64 | 2,61 | 11,87 | 11,88 | 23,75 | 3,64 | 6,45 | 3,05 | 23,31 |
| | 4 | 10,29 | 7,60 | 50,80 | 4,59 | 4,58 | 12,41 | 49,09 | 6,58 | 9,47 | 12,72 | 40,35 | 7,95 | 20,11 | 8,66 | 41,99 |
| 100x10 | 2 | 2,00 | 4,47 | 1,89 | 0,39 | 0,56 | 4,50 | 1,89 | 0,55 | 2,36 | 2,96 | 2,31 | 0,70 | 0,47 | 0,27 | 2,37 |
| | 3 | 3,37 | 7,18 | 14,36 | 1,07 | 1,55 | 6,89 | 13,69 | 1,42 | 5,20 | 5,61 | 13,31 | 1,75 | 1,23 | 0,83 | 13,58 |
| | 4 | 5,55 | 9,13 | 20,86 | 1,86 | 3,42 | 11,63 | 21,00 | 2,59 | 8,69 | 9,09 | 20,89 | 3,10 | 2,77 | 1,96 | 20,89 |
| | 5 | 9,13 | 13,84 | 8,70 | 3,16 | 4,94 | 13,41 | 9,90 | 3,93 | 13,93 | 12,05 | 12,76 | 4,75 | 6,68 | 3,40 | 13,27 |
| | 6 | 11,80 | 14,09 | 27,24 | 4,52 | 5,09 | 11,25 | 28,61 | 5,41 | 15,02 | 13,60 | 30,18 | 6,81 | 11,17 | 5,51 | 30,00 |
| | 7 | 10,44 | 10,60 | 43,88 | 6,23 | 7,83 | 10,82 | 44,30 | 8,01 | 13,45 | 14,57 | 42,98 | 9,76 | 16,14 | 10,01 | 42,24 |
| | 8 | 7,85 | 7,95 | 53,16 | 7,95 | 9,45 | 11,61 | 49,98 | 10,76 | 15,20 | 17,68 | 41,99 | 14,39 | 28,12 | 13,70 | 40,02 |
| | 9 | 9,87 | 10,02 | 29,17 | 10,73 | 12,99 | 14,40 | 24,89 | 14,17 | 22,57 | 23,83 | 25,34 | 22,62 | 43,22 | 16,78 | 25,95 |
| 100x20 | 2 | 0,66 | 1,69 | 0,93 | 0,52 | 0,59 | 1,60 | 1,06 | 0,46 | 1,05 | 1,26 | 1,11 | 0,57 | 0,23 | 0,15 | 1,11 |
| | 3 | 1,75 | 2,63 | 5,39 | 0,94 | 1,22 | 3,24 | 5,27 | 1,01 | 9,64 | 6,46 | 12,99 | 4,48 | 0,62 | 0,51 | 5,64 |
| | 4 | 3,19 | 4,68 | 4,66 | 1,53 | 2,41 | 3,58 | 5,48 | 1,69 | 4,56 | 3,89 | 5,56 | 2,06 | 1,16 | 0,95 | 5,43 |
| | 5 | 5,42 | 4,45 | 7,31 | 2,06 | 3,75 | 6,37 | 7,96 | 2,85 | 6,60 | 5,57 | 8,36 | 3,01 | 2,24 | 1,69 | 7,90 |
| | 6 | 5,58 | 6,87 | 16,21 | 3,05 | 5,14 | 9,35 | 15,06 | 3,39 | 8,86 | 7,36 | 14,70 | 4,04 | 3,46 | 2,60 | 14,54 |
| | 7 | 8,07 | 9,57 | 13,83 | 3,98 | 5,60 | 7,87 | 15,20 | 4,33 | 10,69 | 9,24 | 16,29 | 5,23 | 4,71 | 3,52 | 16,03 |
| | 8 | 10,14 | 11,30 | 24,12 | 4,65 | 7,10 | 10,63 | 23,79 | 5,74 | 13,98 | 11,43 | 23,62 | 6,54 | 6,28 | 5,15 | 22,92 |
| | 9 | 14,62 | 11,89 | 17,96 | 5,67 | 9,66 | 16,33 | 16,97 | 6,96 | 17,64 | 12,96 | 16,88 | 8,00 | 8,95 | 6,58 | 16,40 |
| | 10 | 18,42 | 13,04 | 16,32 | 7,07 | 13,48 | 15,87 | 20,21 | 8,37 | 20,13 | 14,20 | 21,85 | 9,61 | 12,63 | 8,91 | 21,44 |
| | 11 | 18,22 | 15,80 | 26,16 | 8,29 | 11,94 | 16,09 | 29,67 | 9,68 | 21,76 | 15,70 | 30,88 | 11,46 | 15,87 | 11,61 | 30,32 |
| | 12 | 19,83 | 15,30 | 35,15 | 10,11 | 13,65 | 13,81 | 38,23 | 11,42 | 21,41 | 17,43 | 38,76 | 13,53 | 17,42 | 14,19 | 38,14 |
| | 13 | 14,67 | 13,57 | 43,06 | 11,32 | 14,34 | 15,52 | 45,22 | 13,45 | 21,00 | 19,65 | 44,56 | 16,00 | 21,16 | 17,00 | 44,00 |
| | 14 | 15,08 | 14,13 | 50,00 | 13,10 | 16,79 | 16,98 | 48,92 | 15,41 | 21,74 | 22,61 | 47,11 | 19,03 | 25,77 | 20,00 | 45,39 |
| | 15 | 14,92 | 16,59 | 54,07 | 15,10 | 18,74 | 19,00 | 48,11 | 18,00 | 24,30 | 26,12 | 44,31 | 22,87 | 31,93 | 22,47 | 42,57 |
| | 16 | 16,63 | 16,74 | 51,11 | 17,61 | 20,93 | 21,99 | 39,76 | 20,81 | 28,80 | 30,48 | 35,59 | 28,34 | 40,06 | 25,35 | 34,24 |
| | 17 | 18,32 | 18,90 | 37,62 | 20,51 | 24,75 | 26,12 | 28,31 | 24,98 | 34,98 | 35,94 | 35,44 | 35,03 | 48,40 | 28,83 | 35,49 |
| | 18 | 22,06 | 22,76 | 25,68 | 23,33 | 30,10 | 31,02 | 30,29 | 30,59 | 42,26 | 42,58 | 42,31 | 42,33 | 57,86 | 33,53 | 42,73 |
| | 19 | 27,07 | 27,25 | 27,41 | 27,41 | 36,94 | 37,09 | 36,96 | 37,00 | 50,01 | 50,05 | 50,02 | 50,02 | 66,51 | 40,15 | 50,51 |
| 200x10 | 2 | 2,90 | 4,16 | 1,36 | 0,23 | 0,53 | 4,62 | 1,51 | 0,27 | 3,60 | 3,74 | 1,80 | 0,40 | 0,26 | 0,25 | 1,82 |
| | 3 | 7,31 | 7,48 | 16,99 | 0,67 | 1,46 | 8,66 | 16,86 | 0,90 | 7,10 | 7,26 | 15,83 | 1,08 | 0,91 | 0,67 | 15,38 |
| | 4 | 8,07 | 11,73 | 21,92 | 1,23 | 3,24 | 10,99 | 21,86 | 1,59 | 9,69 | 10,43 | 22,32 | 1,95 | 1,89 | 1,12 | 22,42 |
| | 5 | 17,14 | 15,05 | 5,61 | 1,98 | 4,72 | 19,84 | 6,97 | 2,38 | 17,56 | 14,97 | 9,20 | 2,96 | 4,25 | 2,21 | 9,60 |
| | 6 | 24,43 | 20,82 | 24,94 | 2,82 | 4,89 | 22,17 | 26,35 | 3,42 | 22,84 | 16,26 | 27,80 | 4,23 | 7,78 | 4,32 | 28,26 |
| | 7 | 15,28 | 17,33 | 43,71 | 3,93 | 5,04 | 16,98 | 44,66 | 5,03 | 15,64 | 13,47 | 44,43 | 5,99 | 11,75 | 7,34 | 44,70 |
| | 8 | 7,58 | 13,41 | 59,44 | 5,12 | 5,59 | 7,54 | 58,11 | 7,25 | 9,91 | 12,29 | 50,25 | 8,70 | 21,29 | 9,81 | 48,81 |
| | 9 | 5,38 | 8,56 | 45,52 | 6,95 | 6,71 | 7,65 | 37,82 | 10,00 | 13,67 | 15,16 | 26,84 | 13,64 | 36,05 | 9,61 | 24,59 |
| 200x20 | 2 | 1,57 | 0,94 | 0,79 | 0,21 | 1,15 | 1,25 | 0,78 | 0,21 | 1,68 | 1,46 | 0,89 | 0,29 | 0,15 | 0,06 | 0,81 |
| | 3 | 3,02 | 1,95 | 5,29 | 0,54 | 2,32 | 4,13 | 5,25 | 0,49 | 3,14 | 2,79 | 5,31 | 0,64 | 0,48 | 0,26 | 5,24 |
| | 4 | 4,64 | 4,62 | 3,20 | 0,76 | 3,73 | 4,54 | 3,52 | 0,97 | 5,49 | 4,36 | 4,14 | 1,09 | 0,85 | 0,54 | 4,03 |
| | 5 | 6,10 | 5,20 | 3,77 | 1,20 | 5,60 | 6,73 | 4,28 | 1,41 | 7,46 | 6,51 | 5,10 | 1,60 | 1,45 | 0,95 | 5,21 |
| | 6 | 11,10 | 10,51 | 18,35 | 1,78 | 8,43 | 7,98 | 17,40 | 1,88 | 10,49 | 8,41 | 16,87 | 2,18 | 2,10 | 1,51 | 16,65 |
| | 7 | 10,26 | 10,66 | 10,65 | 2,15 | 6,95 | 9,86 | 11,72 | 2,28 | 10,79 | 10,00 | 12,98 | 2,85 | 3,26 | 2,28 | 13,01 |
| | 8 | 8,46 | 11,94 | 23,09 | 2,55 | 7,78 | 15,57 | 23,11 | 3,02 | 10,22 | 12,45 | 23,80 | 3,57 | 4,61 | 3,04 | 23,70 |
| | 9 | 12,68 | 15,00 | 27,49 | 3,15 | 13,36 | 18,54 | 25,11 | 3,69 | 15,12 | 15,20 | 22,66 | 4,35 | 6,17 | 3,72 | 22,86 |
| | 10 | 19,98 | 20,26 | 9,31 | 3,71 | 17,59 | 20,96 | 12,85 | 4,30 | 21,76 | 16,48 | 14,96 | 5,25 | 8,67 | 4,82 | 15,00 |
| | 11 | 26,35 | 17,44 | 19,12 | 4,55 | 21,96 | 17,29 | 22,47 | 5,21 | 26,85 | 16,44 | 24,41 | 6,27 | 11,16 | 6,34 | 24,31 |
| | 12 | 30,29 | 24,06 | 28,81 | 5,34 | 18,16 | 18,74 | 31,83 | 6,21 | 28,91 | 15,45 | 33,21 | 7,39 | 13,07 | 8,29 | 33,03 |
| | 13 | 27,62 | 18,89 | 38,38 | 6,15 | 16,14 | 21,41 | 40,35 | 7,31 | 26,52 | 14,79 | 41,05 | 8,76 | 13,98 | 10,17 | 40,53 |
| | 14 | 22,28 | 14,00 | 46,65 | 7,13 | 13,71 | 13,97 | 47,77 | 8,68 | 21,07 | 14,75 | 46,88 | 10,41 | 17,33 | 12,43 | 46,04 |
| | 15 | 12,38 | 10,98 | 53,92 | 8,32 | 12,46 | 11,04 | 52,48 | 10,13 | 16,73 | 16,06 | 49,95 | 12,45 | 23,71 | 13,91 | 49,29 |
| | 16 | 11,46 | 10,67 | 59,01 | 9,58 | 11,50 | 12,40 | 55,04 | 12,21 | 16,54 | 18,18 | 46,62 | 15,16 | 31,29 | 14,62 | 46,53 |
| | 17 | 9,58 | 10,67 | 56,15 | 10,95 | 13,38 | 13,88 | 45,01 | 13,76 | 19,75 | 21,22 | 33,06 | 19,17 | 39,79 | 15,58 | 34,02 |
| | 18 | 10,96 | 11,62 | 36,52 | 12,90 | 15,48 | 16,40 | 23,67 | 15,97 | 24,75 | 25,50 | 25,49 | 24,73 | 47,66 | 17,50 | 25,67 |
| | 19 | 13,35 | 13,76 | 17,50 | 14,66 | 20,35 | 20,59 | 20,27 | 20,48 | 31,00 | 31,19 | 31,01 | 31,04 | 55,54 | 20,60 | 31,14 |
| 500x20 | 2 | 2,04 | 1,23 | 0,36 | 0,09 | 1,16 | 2,32 | 0,42 | 0,09 | 2,05 | 2,18 | 0,44 | 0,13 | 0,11 | 0,07 | 0,40 |
| | 3 | 4,52 | 4,05 | 5,36 | 0,18 | 3,28 | 4,00 | 5,40 | 0,24 | 4,35 | 4,21 | 5,51 | 0,30 | 0,32 | 0,22 | 5,52 |
| | 4 | 7,71 | 6,48 | 1,58 | 0,34 | 4,71 | 5,54 | 2,02 | 0,42 | 6,84 | 6,06 | 2,35 | 0,53 | 0,70 | 0,44 | 2,57 |
| | 5 | 9,96 | 8,55 | 2,29 | 0,52 | 6,95 | 8,61 | 2,85 | 0,67 | 9,09 | 8,39 | 3,37 | 0,80 | 1,00 | 0,73 | 3,57 |
| | 6 | 7,78 | 9,54 | 19,68 | 0,76 | 5,71 | 7,94 | 18,94 | 0,97 | 8,62 | 10,26 | 19,01 | 1,10 | 1,60 | 1,03 | 18,90 |
| | 7 | 16,23 | 12,77 | 8,15 | 1,03 | 10,75 | 14,79 | 9,27 | 1,16 | 16,09 | 13,99 | 9,96 | 1,44 | 2,16 | 1,53 | 10,03 |
| | 8 | 16,97 | 13,87 | 21,91 | 1,29 | 6,39 | 12,35 | 22,37 | 1,48 | 13,84 | 12,38 | 22,86 | 1,81 | 2,75 | 2,06 | 22,99 |
| | 9 | 9,27 | 13,34 | 33,03 | 1,54 | 7,35 | 12,63 | 29,70 | 1,82 | 10,77 | 13,48 | 28,39 | 2,20 | 3,66 | 2,64 | 28,57 |
| | 10 | 17,73 | 15,23 | 4,58 | 1,91 | 13,23 | 19,50 | 6,86 | 2,15 | 18,09 | 19,08 | 9,04 | 2,65 | 5,42 | 3,24 | 9,09 |
| | 11 | 23,99 | 25,08 | 14,50 | 2,27 | 20,07 | 24,24 | 16,61 | 2,54 | 26,06 | 23,77 | 18,72 | 3,14 | 7,20 | 4,21 | 19,08 |
| | 12 | 33,29 | 29,95 | 24,45 | 2,71 | 22,02 | 30,35 | 26,37 | 3,05 | 32,91 | 25,51 | 28,19 | 3,70 | 8,70 | 5,55 | 28,64 |
| | 13 | 38,54 | 31,98 | 34,13 | 3,23 | 25,26 | 32,81 | 35,80 | 3,67 | 37,03 | 23,73 | 37,24 | 4,36 | 9,38 | 7,50 | 37,38 |
| | 14 | 39,19 | 29,37 | 43,74 | 3,75 | 22,99 | 28,06 | 44,75 | 4,46 | 36,44 | 18,77 | 45,52 | 5,14 | 10,99 | 9,57 | 45,84 |
| | 15 | 40,98 | 26,00 | 52,84 | 4,44 | 18,73 | 24,29 | 53,41 | 5,19 | 26,90 | 13,47 | 52,60 | 6,12 | 15,59 | 10,87 | 53,00 |
| | 16 | 18,98 | 14,61 | 60,70 | 5,08 | 11,37 | 12,06 | 60,21 | 6,31 | 13,87 | 11,28 | 56,60 | 7,40 | 22,51 | 11,28 | 56,31 |
| | 17 | 7,78 | 10,00 | 66,73 | 6,03 | 6,77 | 8,76 | 61,87 | 7,78 | 10,36 | 11,57 | 50,73 | 9,15 | 30,37 | 10,60 | 49,33 |
| | 18 | 5,23 | 5,89 | 58,11 | 6,86 | 6,77 | 7,54 | 46,07 | 9,07 | 12,42 | 13,48 | 30,39 | 11,91 | 37,51 | 9,86 | 28,64 |
| | 19 | 5,52 | 5,79 | 31,29 | 8,19 | 8,60 | 8,89 | 15,79 | 9,75 | 16,66 | 17,03 | 17,28 | 16,59 | 45,49 | 10,39 | 17,22 |

**Tableau 5.6: Pourcentages de déviation moyens du makespan des solutions comparé à la borne inférieure $\beta_e(k)$.**

type avec $\sum_{h=1}^{\ell} k_h = k$. Un opérateur de type $\ell$ a un niveau de performance $v_\ell$.

Nous supposons que la séquence de jobs est fixée au départ. Comme nous traitons le problème du flow shop standard [7] avec opérateurs, nous notons ladite séquence $\pi = (\pi_{ij})$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. L'objectif est de trouver une affectation d'opérateurs $a^*$ de façon à minimiser le makespan ; $a^*$ est telle que :

$$C_{\max}(\pi, a^*) = \min_{a \in A} C_{\max}(\pi, a),$$

où $A$ représente l'ensemble des affectations réalisables, $a^*$ est dans $A$, et $C_{max}(\pi, a)$ est le temps de complétion final de la séquence $\pi$ sous l'affectation $a$.

Avant de poursuivre, décrivons la notation utilisée ici :

- Comme les opérations peuvent être interrompues, une opération $\pi(i, j)$ serait composée de $o_{ij}$ sous-opérations : $\pi(i, j)_b$, $b = 1, \ldots, o_{ij}$.
- Quand une opération n'est pas interrompue, $\pi(i, j) = \pi(i, j)_1$.
- Un opérateur $h$, $h = 1, \ldots, k$, traite $n_h$ sous-opérations : $op_c$, $c = 1, \ldots, n_h$.
- $\Delta_i$ est l'ensemble des machines qui précèdent la machine $M_i$ sur la route d'un job $j$, les machines $M_p, p < i$.
- $p_{h,op_c,\pi(i,j)_b}$ est le temps durant lequel l'opérateur $h$ traite sa $c$-ème sous-opération, $1 \leq c \leq n_h$, qui correspond à une opération $\pi(i, j)_b$, $1 \leq i \leq m$, $1 \leq j \leq n$ et $1 \leq b \leq o_{ij}$.
- $av_{h,op_c,\pi(i,j)_b}$ est l'instant auquel la $c$-ème sous-opération traitée par l'opérateur $h$ devient disponible [8].

7. Sans contrainte de permutation.

8. Cette opération correspond à une opération $\pi(i, j)_b$, $1 \leq i \leq m$, $1 \leq j \leq n$ et $1 \leq b \leq o_{ij}$. Une sous-opération devient disponible quand toutes les sous-opérations qui la précèdent ainsi que les opérations du job $j$ sur les machines dans $\Delta_i$ ont été complétées.

— $ts_{h,op_c,\pi(i,j)_b}$ est l'instant auquel l'opérateur $h$ en charge de la sous-opération [9] $\pi(i,j)_b$ commence à la traiter.

— $C(\pi(i,j)_b,a)$ et $C(\pi(i,j),a)$ représentent les dates de complétion de la sous-opération $\pi(i,j)_b$ et de l'opération $\pi(i,j)$ sous l'affectation $a$. Elles sont calculées suivant les formules récursives suivantes.

Étant donnée une solution $(\pi,a)$, les dates de complétion des opérations sont calculées comme suit.

$$
\begin{aligned}
ts_{h,op_0,\pi(i,j)_b} &= av_{h,op_0,\pi(i,j)_b} = p_{h,op_0,\pi(i,j)_b} = -\infty;\ h = 1,\ldots,k. \\
ts_{h,op_c,\pi(i,j)_b} &= \max\left\{0, ts_{h,op_{c-1},\pi(i',j')_{b'}} + \frac{p_{h,op_{c-1},\pi(i',j')_{b'}}}{v_h}, av_{h,op_c,\pi(i,j)_b}\right\}; \\
&\quad h = 1,\ldots,k;\ c = 1,\ldots,n_h. \qquad (5.9) \\
av_{h,op_c,\pi(i,j)_b} &= \max\left\{0, \max_{i'\in\Delta_{ij}} C(\pi(i',j),a), C(\pi(i,j)_{b-1},a)\right\}; \\
&\quad h = 1,\ldots,k;\ c = 1,\ldots,n_h. \qquad (5.10) \\
C(\pi(i,j)_b,a) &= ts_{h,op_c,\pi(i,j)_b} + \frac{p_{h,op_c,\pi(i,j)_b}}{v_h}; \\
&\quad i = 1,\ldots,m;\ j = 1,\ldots,n;\ b = 1,\ldots,o_{ij}. \qquad (5.11) \\
C(\pi(i,j),a) &= C(\pi(i,j)_{o_{ij}},a);\ i = 1,\ldots,m;\ j = 1,\ldots,n. \qquad (5.12)
\end{aligned}
$$

Les équations (5.9) calculent les temps auxquels les opérateurs commencent le traitement des opérations leur étant affectées. Pour chaque opérateur, ces temps de début correspondent soit au temps où l'opération en question devient disponible ou au temps où l'opérateur termine l'opération qui la précède ou au temps 0 pour leur première opération.

9. qui est sa $c$-ème sous-opération.

Les équations (5.10) calculent les dates de disponibilité des opérations. Une opération $S(i,j)$ devient disponible lorsque toutes les opérations la précédant sur la machine $M_i$ ainsi que les opérations du job $j$ sur les machines $M_{i'}$, $i' < i$, ont été complétées.

Finalement, les équations (5.11) et (5.12) calculent les temps de complétion des opérations.

Comme pour chaque problème d'optimisation combinatoire, nous commençons par l'étude de complexité. Dans ce qui suit, nous notons notre problème $F_m|res\ \ell k_h 1, S, v_h, f|C_{\max}$ selon la notation présentée à la Section 5.2. Ainsi, nous étudions le problème du flow shop à $n$ jobs et $m$ machines avec $\ell$ types d'opérateurs, $k_h$, $h = 1, \ldots, \ell$ opérateurs de chaque type avec $\sum_{h=1}^{\ell} k_h = k$, et une séquence donnée de jobs notée $S$. L'objectif est la minimisation du makespan.

**A. Le cas avec $k$ arbitraire**

Dans ce qui suit, nous démontrons un résultat de complexité pour le cas où le nombre d'opérateurs $k$ est arbitraire.

Notre preuve consiste à montrer que le problème de décision suivant est $\mathscr{NP}$-complet pour $k$ arbitraire. $FSMS(m)$ est un problème de flow shop avec $\ell$ jobs, $m \geq 2$ machines et un nombre arbitraire $k$ d'opérateurs où le changement d'affectation des opérateurs se fait selon le mode libre et $S$ est une séquence fixée de jobs. Nous supposons aussi que les opérateurs ont le même niveau de performance, et que l'objectif est la minimisation du makespan. Étant donné un entier $\alpha$, existe-t-il une affectation $a \in A$ telle que $C_{max}(S,a) \leq \alpha$ ?

La réduction est construite à partir du problème à machines parallèles suivant qui est connu pour être $\mathscr{NP}$-complet [Ullman (1976)].

*PM* est un problème à machines parallèles avec $n$ jobs et un système de contraintes de précédence quelconque, où $p_h$ est le temps de traitement du job $h$, $1 \leq h \leq n$. Les préemptions sont autorisées. Étant donné un entier $\beta$, existe-t-il un ordonnancement $\pi$ tel que $C_{max}(\pi) \leq \beta$ ?

Soit $I$ une instance du problème *PM*. Nous construisons une instance $I'$ de $FSMS(m)$ comme suit. À partir du graphe de précédence $G$ de $I$, nous construisons le graphe de précédence $G'$ de $I'$ comme suit :

— Chaque sommet dans $G$ correspond à une opération d'un job dans $G'$. Les opérations dans $G'$ ont les mêmes temps de traitement que les jobs qui leur correspondent dans $G$. Dans $I'$, nous considérons des jobs avec des opérations manquantes.

— Nous choisissons dans $G$ un job sans aucun prédécesseur, la racine, ce sera le premier job sur la première machine dans $G'$.

— Supposons qu'un job $j$ a $d$ successeurs, $d \geq 3$, alors on ajoute $\left\lceil \frac{d}{2} \right\rceil$ sommets artificiels, chacun aura un degré intérieur de 1 (car il aura un arc venant du sommet associé au job $j$) et un degré extérieur d'au plus 2 (chaque sommet artificiel aura pour successeurs au plus deux successeurs du sommet associé au job $j$. Les liens sont fait de façon à ce que chaque successeur de $j$ ait exactement un sommet artificiel pour prédécesseur). Nous répétons cette procédure jusqu'à ce que chaque sommet ait au plus deux successeurs, un de gauche et un de droite. Les sommets artificiels correspondent à des opérations qui ont un temps de traitement égal à 0. La construction du graphe $G'$ est illustrée en Figure 5.11 pour le cas des successeurs, nous procédons de manière analogue pour les prédécesseurs.

— Supposons qu'un job $j$ a $d$ prédécesseurs, $d \geq 3$ alors nous ajoutons $\left\lceil \frac{d}{2} \right\rceil$ sommets

artificiels, chacun aura un degré extérieur de 1 (car il aura un arc vers le sommet associé au job $j$) et un degré intérieur d'au plus 2 (chaque sommet artificiel aura pour prédécesseurs au plus deux prédécesseurs du sommet associé au job $j$. Les liens sont fait de façon à ce que chaque prédécesseur de $j$ ait exactement un sommet artificiel pour successeur). Nous répétons cette procédure jusqu'à ce que chaque sommet ait au plus deux prédécesseurs. Les sommets artificiels correspondent à des opérations qui ont un temps de traitement égal à 0.

— $n$, le nombre de jobs de $I'$, est la longueur du chemin qui va de la racine au dernier sommet à n'avoir pas de successeur droit suivant un parcours préfixe dans le sous-graphe droit, soit $F$ ce chemin. Pour chaque opération, son successeur gauche représenterait l'opération du même job sur la machine suivante alors que son successeur droit représenterait l'opération du prochain job sur la même machine.

— $m$, le nombre de machines de $I'$, est la longueur du plus long des chemins qui partent d'un sommet de $F$ en prenant toujours l'enfant de gauche jusqu'à arriver à un sommet sans enfant gauche.

— $\alpha = \beta$

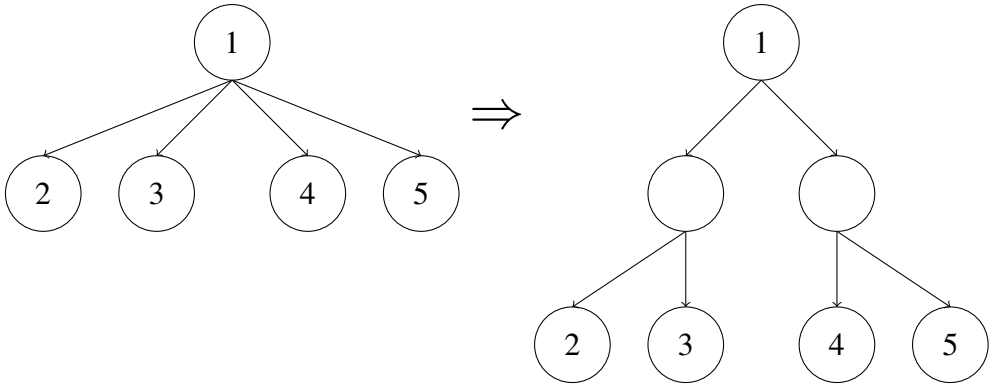



**Figure 5.11: Exemple de construction du graphe $G'$.**

Il est clair que cette construction peut se faire en temps polynomial. Maintenant, nous montrons

que l'instance $I$ a la réponse "oui" si, et seulement si, $I'$ a la réponse "oui".

**Lemme 9.** *Si $I$ a la réponse "oui", alors $I'$ a la réponse "oui".*

*Démonstration.* Si $I$ a la réponse "oui", nous avons un ordonnancement $\pi$ tel que $C_{max}(\pi) \leq \beta$. Nous pourrions ajouter autant de jobs ayant un temps de traitement égal à 0 que nous voudrions sans augmenter le makespan. Les machines de $I$ correspondent aux opérateurs de $I'$. Ainsi, nous pourrions construire une solution de $I'$ avec $C_{max}(S,a) \leq \alpha$ en ajoutant les opérations correspondant aux sommets artificiels. Une telle solution respecterait les contraintes de précédences de $G'$. Ainsi, $I'$ aura la réponse "oui". □

**Lemme 10.** *Si $I'$ a la réponse "oui", alors $I$ a la réponse "oui".*

*Démonstration.* Si $I'$ a la réponse "oui", nous avons une affectation d'opérateurs $a$ et une séquence $S$ tels que $C_{max}(S,a) \leq \alpha$. Comme les opérateurs de $I'$ correspondent aux machines de $I$, à partir de la solution de $I'$, nous pourrions construire un ordonnancement $\pi$ pour $I$ en sautant les opérations ayant un temps de traitement égal à 0 tel que $C_{max}(\pi) \leq \beta$. Comme les contraintes de précédence de $I'$ incluent celles de $I$, alors un tel ordonnancement respectera les contraintes de précédence de cette dernière. Ainsi, $I$ a la réponse "oui". □

**Théorème 4.** *Le problème $F_m|res\ \ell \cdot 1, S, f|C_{max}$ est $\mathscr{NP}$-difficile pour un nombre k arbitraire d'opérateurs.*

*Démonstration.* Le problème $FSMS(m)$ est clairement dans $\mathscr{NP}$. Par les Lemmes 9 et 10, l'instance $I$ de $PM$ a une solution si, et seulement si, l'instance $I'$ de $FSMS(m)$ a une solution. Le résultat du théorème est donc établi. □

**Corollaire 1.** *Le problème $F_m|res\ \ell \cdot 1, S, v_h, f|C_{\max}$ est $\mathscr{NP}$-difficile pour un nombre arbitraire d'opérateurs.*

*Démonstration.* Évidente car $F_m|res\ \ell \cdot 1, S, f|L_{max}$ est un cas spécial de $F_m|res\ \ell \cdot 1, S, v_h, f|L_{\max}$ où tous les opérateurs ont les mêmes niveaux de performance. □

**B. Le cas avec $k = 2$ opérateurs**

**Théorème 5.** *Le problème $F_m|res\ 2k_h 1, S, v_h, f|C_{\max}$ est résoluble en temps polynomial.*

*Démonstration.* La preuve de complexité est faite via la méthode de restriction. Nous notons d'abord que les contraintes d'ordre imposées par la séquence $S$ et par la route de chaque job forment un graphe de précédence. Ainsi, notre problème peut être vu comme un problème à machines parallèles uniformes avec contraintes de précédence et possibilité de préemption, noté $Q_2|prec, prmp|C_{\max}$, où les machines, les jobs et le graphe de précédence représentent respectivement les 2 opérateurs, les $m * n$ opérations, et les relations de précédence entre ces dernières. Le problème $Q_2|prec, prmp|C_{\max}$ est résoluble en temps polynomial. En effet,

$$Q_2|prec, prmp|C_{\max} \propto Q_2|prec, prmp, r_j|L_{\max}$$

et $Q_2|prec, prmp, r_j|L_{\max}$ est résoluble en temps polynomial [Lawler (1982)]. □

Nous présentons maintenant une méthode de résolution pour le cas à deux opérateurs.

Le *niveau* $L_s(T_{S(i,j)})$ d'un job [10] $T_{S(i,j)}$ à un temps $s$ est le temps minimum nécessaire pour le traitement dudit job ainsi que tous ses successeurs à partir du temps $s$. Pour notre problème, $L_s(T_{S(i,j)}) = \left( \sum_{x=i}^{m} \sum_{y=j}^{n} p_{x,S(x,y)} \right) - p_{i,S(i,j)}$.

L'algorithme que nous présentons en Figure 5.12 est en $O(m^2 n^2)$ est une adaptation de l'algorithme de [Horvath et al. (1977)]. Il fourni une solution avec une makespan minimum

10. Le job à la $j^{me}$ position sur la $i^{me}$ machine dans la séquence $S$.

pour $k = 2$ dans le cas où les opérateurs sont identiques mais aussi quand ils ont des niveaux de performance différents. En pratique, nous aurions d'abord à affecter des valeurs aux vitesses respectives des opérateurs en se basant sur leurs niveaux de performance avant l'application de l'algorithme.

$s \leftarrow 0$ ;
$h \leftarrow$ le nombre d d'opérateurs libres au temps $s$ ;
$j \leftarrow$ le nombre de jobs de plus haut niveau au temps $s$ ;
**Faire**
{
**Si**($j \leq h$)
{ Affecter les $j$ opérateurs les plus rapides aux $j$ jobs ; }
**Sinon**
{ Les $j$ jobs sont exécutés au même taux par les $h$ opérateurs($^*$) ; }
**Si**( $\exists$ opérateurs libres)
{Affecter les jobs du prochain plus haut niveau ; }
**Si**(( $\exists$ $T$ qui se termine à $t$) ou ($\exists$ $T$, $T'$ / $L_s(T) > L_s(T')$ et $L_t(T) = L_t(T')$))
{ $s \leftarrow t$ ; }
} **Tant que**(Tous les jobs n'ont pas été exécutés)
Pour construire une solution respectant le mode de changement d'affectation libre, les $j$ jobs qui se partagent les $h$ opérateurs durant les étapes ($^*$) reçoivent le même temps de traitement des dits opérateurs ;
Chaque intervalle partagé est divisé en $j$ sous-intervalles ;
Chacun des $j$ jobs est ordonnancé dans $h$ sous-intervalles, à chaque fois avec un opérateur différent ;

**Figure 5.12: Pseudo-code d'un algorithme pour la résolution du problème $F_m|res\ 2k_h1, S, v_h, f|C_{max}$.**

### C. Le cas avec $k = 1$ opérateur

Bien que le cas à un opérateur soit un cas particulier de celui à deux opérateurs, plutôt que d'utiliser une réduction de ce dernier, nous présentons dans ce qui suit un algorithme plus simple pour sa résolution .

**Théorème 6.** *Le problème $F_m|res\ 111, S, f|C_{\max}$ est résoluble en temps polynomial.*

*Démonstration.* Nous utilisons la même technique de preuve que celle pour $k = 2$. Nous utilisons pour la réduction le problème à machine unique $1|prec, prmp|C_{\max}$ qui est résoluble en temps polynomial. En effet,

$$1|prec, prmp|C_{max} \propto 1|prec, prmp, r_j|L_{max}$$

et $1|prec, prmp, r_j|L_{\max}$ est résoluble en temps polynomial [Baker et al. (1983)]. □

Comme nous l'avons précédemment mentionné, le concept à un opérateur et plusieurs machines (OWMM (One Worker Multiple Machine)) est très populaire, particulièrement dans les systèmes de production juste-à-temps ; il est souvent rencontré en pratique.

Si $k = 1$, nous remarquons que la somme de tous les temps de traitement $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$ est une borne inférieure évidente. Nous notons que chaque solution *sans retard* [11] a un makespan égal à cette borne inférieure. Son optimalité est ainsi établie. Une telle solution peut être construite en $O(mn)$.

Une politique d'ordonnancement optimale serait de traiter, sans interruption, tous les jobs de la première machine, ensuite tous ceux de la deuxième machine et ainsi de suite jusqu'à la machine $M_m$.

**D. Conclusion**

Dans cette section, nous avons étudié l'affectation d'un nombre d'opérateurs inférieur au nombre de machines dans un environnement de type flow shop avec une séquence de jobs donnée et un mode de changement d'affectation libre pour la minimisation du makespan. Nous

11. Une solution où aucune machine ne reste inactive tant qu'il y a des jobs disponibles pour être traités par elle et qu'il y a un opérateur disponible pour la faire fonctionner.

avons étudié la complexité de notre problème dans le cas où les opérateurs ont des niveaux de performance différents. Ce problème est $\mathscr{NP}$-difficile au sens fort, sauf pour les cas à un et deux opérateurs pour lesquels nous avons présenté des algorithmes de résolution en temps polynomial. Étant donné la $\mathscr{NP}$-difficulté au sens fort du problème quand $k \geq 3$, il serait intéressant d'explorer la piste des méthodes heuristiques. Par ailleurs, il serait aussi intéressant de tester les algorithmes présentés sur des instances réelles. Nous conseillons aussi l'étude d'autres objectifs comme la somme de temps de fin de traitement.

### *5.4.2 MINIMISATION DU RETARD ALGÉBRIQUE MAXIMUM DANS LE CAS SÉQUENTIEL*

Dans la présente section, nous étudions la minimisation du $L_{\max}$, dénotant le retard algébrique maximum des jobs lors de l'accomplissement d'un ordonnancement donné. Pour rendre le modèle plus réaliste, nous supposons que les opérateurs ont des niveaux de performance différents. Nous démontrons la $\mathscr{NP}$-difficulté du problème quand le nombre d'opérateurs est arbitraire et présentons des algorithmes polynomiaux pour les cas à un et deux opérateurs. Les résultats ont été présentés dans [Benkalai et al. (2016a)].

#### A. Description du problème

Dans le présent modèle, nous supposons qu'une opération nécessite la présence de l'un des $k$ opérateurs ($k < m$) pour toute la durée de son traitement. Ainsi, l'impact du partage d'opérateurs est modélisé par des temps d'attente engendrés par le fait que le nombre d'opérateurs est inférieur au nombre de machines. Les paramètres $p_{ij}$, $d_{ij}$ et $C_{ij}$ représentent respectivement le temps de traitement, la date d'échéance et la date de fin de traitement du job $j$ sur la $i^{me}$ machine.

Nous avons $\ell$ types d'opérateurs, $k_h$, $h = 1, \ldots, \ell$, opérateurs de chaque type avec $\sum_{h=1}^{\ell} k_h = k$. Un opérateur de type $h$ a une cadence $v_h$. Le changement d'affectation de ces opérateurs aux différentes machines se fait suivant le mode libre. Autrement dit, un opérateur pourra interrompre le traitement d'un job à tout instant pour s'occuper d'un autre job. Ce job pourra par la suite être repris par ce même opérateur ou être confié à un autre opérateur. L'objectif est de minimiser le retard algébrique maximum, $L_{\max} = \max_{j=1,\ldots,n} L_{mj}$ où $L_{ij} = C_{ij} - d_{ij}$.

Dans ce qui suit, nous présentons l'étude de complexité de nos problèmes ainsi que des méthodes de résolution en temps polynomial pour les cas faciles.

Nous notons notre problème $F_m|res\ \ell k_h 1, S, v_h, f|L_{\max}$ conformément à la notation présentée à la Section 5.2.

**B. Le cas avec $k$ arbitraire**

**Théorème 7.** *Le problème $F_m|res\ \ell \cdot 1, S, f|L_{max}$ est $\mathscr{N}\mathscr{P}$-difficile pour un nombre arbitraire d'opérateurs.*

*Démonstration.* Évidente car $C_{max}$ est un cas spécial de $L_{max}$ où toutes les dates d'échéance sont égales à 0, et le problème pour la minimisation du $C_{max}$ est $\mathscr{N}\mathscr{P}$-difficile, voir Section 5.4.1. □

**Corollaire 2.** *Le problème $F_m|res\ \ell \cdot 1, S, v_h, f|L_{\max}$ est $\mathscr{N}\mathscr{P}$-difficile pour un nombre arbitraire d'opérateurs.*

*Démonstration.* Évidente car $F_m|res\ \ell \cdot 1, S, f|L_{max}$ est un cas spécial de $F_m|res\ \ell \cdot 1, S, v_h, f|L_{\max}$ où tous les opérateurs ont les mêmes niveaux de performance. □

### C. Le cas avec $k = 2$ opérateurs

**Théorème 8.** *Le problème $F_m|res\ 2k_h 1, S, v_h|L_{\max}$ est résoluble en temps polynomial.*

*Démonstration.* Nous utilisons le même principe de preuve que présenté dans le cas du $C_{max}$ (Voir Section 5.4.1). Nous utilisons pour la réduction $Q_2|prec, prmp, r_j|L_{\max}$ qui est résoluble en temps polynomial [Lawler (1982)]. □

La solution est construite en trois étapes. Les dates d'échéance sont d'abord modifiées (Figure 5.13), des intervalles variables sont ensuite construits (Figure 5.14). Enfin, une solution est obtenue avec un algorithme d'ordonnancement par priorité en $O(m^2n^2)$ qui est présenté en Figure 5.15. Ces algorithmes sont des adaptations de ceux de Lawler [Lawler (1982)]. Ils permettent de trouver une solution avec un $L_{\max}$ minimum pour $k = 2$ dans le cas où les opérateurs sont identiques et aussi lorsqu'ils possèdent des niveaux différents de performance.

Soient les données suivantes :

- $d_{ij} = d_j - \sum_{q=m}^{i+1} p_{qj}$ la date d'échéance de la $i^{me}$ opération du job $j$.
- Sans perdre de généralité, $v_1 = 1$, $v_2 = \frac{min\{v_1,v_2\}}{max\{v_1,v_2\}}$.
- $p_{ij}(t)$ le temps de traitement minimum qui doit être effectué sur l'opération $(i, j)$ avant la date $t$ pour que ladite opération respecte sa date d'échéance.
- $S(i, j)$ l'ensemble des successeurs (immédiats ou pas) de l'opération $(i, j)$.

Pour classer les opérations par ordre de priorité, définissons d'abord $b_{ij}^{(r)} = d_{ij} - p_{ij}^{(r)}$, où $p_{ij}^{(r)}$ est le temps de traitement restant à effectuer sur l'opération $(i, j)$ à l'instant $t_r$. Plus $b_{ij}^{(r)}$ est petit, plus la priorité de l'opération $(i, j)$ est grande. Si à un instant $t_r$ il y a $z$ opérations disponibles, elles seront ré-indexées selon l'ordre croissant des $b_{ij}^{(r)}$.

Soit $x_{ij}^{(r)}$ la quantité de l'opération $(i, j)$ exécutée durant l'intervalle $[t_r, t_{r+1}]$.

$$x_{ij}^{(r)} = \begin{cases} \Delta, & q < u, \\ max\{0, T - b_q^{(r)}\}, & q \geq u, \end{cases}$$

avec $T$, $u$ et $v$ déterminés par l'algorithme en Figure 5.14, $v$ étant l'indice tel que $b_v^{(r)} \leq T < b_{v+1}^{(r)}$ et $q$ l'indice correspondant à l'opération $(i, j)$ après ré-indexation.

**Pour** (toute opération $(i, j)$ sans successeur)
$\{d'_{ij} \leftarrow d_{ij}\,;\}$
Créer deux listes ordonnées par date d'échéance modifiées $d'_{ij}$ et $b'_{ij} = d'_{ij} - p_{ij}$ ;
**Tant que** ($\exists$ $(i, j)$ / $d_{ij}$ non modifiée et $d_{i'j'}$ modifiée $\forall (i', j') \in S(i, j)$)
{ Choisir $(i, j)$ ;
Chercher les successeurs de cette opération dans les deux listes ordonnées ;
Calculer $\sum_{(i',j') \in S(i,j)} p_{i'j'}(d'_{i''j''})\ \forall (i'', j'') \in S(i, j)$ ;

$$d'_{ij} \leftarrow min \left\{ d_{ij}, \min_{d'_{(i'',j'')} / (i'',j'') \in S(i,j)} \left\{ d'_{(i'',j'')} - \frac{\sum_{(i',j') \in S(i,j)} p_{i'j'}(d'_{i''j''})}{v_1 + v_2} \right\} \right\} ;$$

**Pour** (chaque $(i', j') \in S(i, j)$)
$\{d'_{ij} \leftarrow min\{d'_{ij}, b'_{i'j'} = d'_{i'j'} - p_{i'j'}\}\,;\}$
Insérer $d'_{ij}$ et $b'_{ij} = d'_{ij} - p_{ij}$ dans les listes ordonnées ;
}

**Figure 5.13: Algorithme pour la modification des dates d'échéance.**

$\Delta \leftarrow 0$ ; $u \leftarrow m$ ; $v \leftarrow m$ ; $T \leftarrow b_m$ ; $P \leftarrow 0$ ;
**Tant que** $(\Delta < t_{r+1} - t_r)$
{
  **Tant que** $(T = b_{u-1} + \Delta)$
  { $u \leftarrow u-1$ (Si $u = 1$, $b_0 = -\infty$) ; $P \leftarrow P + \Delta$ ;}
  **Tant que** $(T = b_{v+1})$
  { $v \leftarrow v+1$ (Si $v = z$, $b_{z+1} = \infty$) ;}
  $T_1 \leftarrow (\frac{(b_{u-1}+\Delta)(m-u+s)-(v-u+1)T)}{((m-u+s)-(v-u+1))}$ ; $T_2 \leftarrow b_{v+1}$ ; $T_3 \leftarrow T + \frac{(t_{r+1}-t_r-\Delta)(m-u+s)}{(v-u+1)}$ ;
  $T' \leftarrow min\{T_1, T_2, T_3\}$ ; $P \leftarrow P + (v-u+1)(T'-T)$ ; $\Delta \leftarrow \frac{P}{(m-u+s)}$ ; $T \leftarrow T'$ ;
}

**Figure 5.14: Algorithme pour la construction d'intervalles fixes.**

Calculer le degré intérieur (nombre de prédécesseurs) de chaque opération $(i, j)$ ;
Créer une file $Q$ contenant les opérations de degré intérieur égal à 0 ;
Créer une liste $A$ contenant les opérations disponibles pour traitement* ;
$r \leftarrow 0$ ;
**Tant que** $(Q \neq \varnothing)$
{ $r \leftarrow r+1$ ; $t_r \leftarrow 0$ ; $A \leftarrow A \cup Q$ ; $Q \leftarrow \varnothing$ ;
  **Tant que** $(A \neq \varnothing)$
  { $t_{r+1} \leftarrow 0$ ;
    Exécuter l'algorithme en Figure 5.14 sur les opérations dans $A$ ;
    **Si** (les valeurs trouvées $(\Delta, u, v, T)$ violent $b_j + \Delta \leq d_j$, $j = 1, \ldots, u-1$)
    { Ré-exécuter l'algorithme en Figure 5.14 avec $t_{r+1} = t_r + \min_y \{p_j^y / j = 1, \ldots, u-1\}$ ;
      $A \leftarrow A \backslash \{$opérations complétées dans$[t_r, t_{r+1}]\}$ ;
      Mettre à jour les degrés intérieurs des opérations ;
      Mettre à jour $Q$ ; $A \leftarrow A \cup Q$ ; $k \leftarrow k+1$ ;
    }
  }
}

N.B : Pour la construction d'intervalles variables, on procède de manière analogue à la différence des points suivants :

— $T \leq d_j$, $j = u, u+1, \ldots, v$. Si $T = d_j$, $j \geq u$, le calcul s'arrête.

— Si $u$ est décrémenté, on vérifie si $b_{u-1} + \Delta > d_{u-1}$, si c'est le cas, le calcul s'arrête.

* Une opération est dite disponible à la date $t$ si le traitement de tous ses prédécesseurs est complété et s'il reste un temps de traitement non nul à effectuer sur cette opération.

**Figure 5.15: Algorithme pour la résolution de** $F_m|res\ 2k_h 1, S, v_h|L_{\max}$**.**

### D. Le cas avec $k = 1$ opérateur

Ici encore, bien que le cas à un opérateur soit un cas particulier de celui à deux opérateurs, plutôt que d'utiliser une réduction de ce dernier, nous présentons dans ce qui suit un algorithme plus simple pour sa résolution .

**Théorème 9.** *Le problème* $F_m|res\ 111, S|L_{\max}$ *est résoluble en temps polynomial.*

*Démonstration.* Nous utilisons le même principe de preuve que précédemment. Nous utilisons pour la réduction $1|prec, prmp, r_j|L_{\max}$ qui est résoluble en temps polynomial [Baker et al. (1983)]. □

L'algorithme présenté en Figure 5.16 est une adaptation de l'algorithme de [Baker et al. (1983)]. Son temps d'exécution est en $O(m^2n^2)$. Étant donné que toutes les dates de disponibilités ($r_j$) sont égales à 0, toutes les opérations appartiennent à un même block $B$ et peuvent être traitées sans temps d'arrêt. Soit $P(B) = \sum_{(i,j)\in B} p_{ij}$.

**Tant que** $(B \neq \varnothing)$
{ Choisir $(i, j)$ / $S(i, j) = \varnothing$ et $L_{ij} = \min_{(i,j)\in B} \{P(B) - d_{ij}\}$ ;
$B \leftarrow B \backslash \{(i, j)\}$ ;
}

**Figure 5.16: Algorithme pour la résolution de** $F_m|res\ 111, S|L_{\max}$**.**

### E. Conclusion

Dans la présente Section, nous avons étudié l'affectation d'opérateurs ayant des niveaux différents de performances dans un environnement de type flow shop. Le nombre d'opérateurs est inférieur à celui des machines, l'ordre des tâches est donné, le mode de changement

d'affectation est libre et l'objectif est la minimisation du retard algébrique maximum. Ce problème s'est avéré être NP-difficile au sens fort, excepté pour les cas à un et deux opérateurs pour lesquels nous avons fourni une méthode de complexité polynomiale. Il serait intéressant de penser au développement de méthodes heuristiques pour la résolution des cas avec $k \geq 3$ opérateurs, mais aussi de considérer d'autres objectifs importants en pratique tels que que le temps moyen d'accomplissement des jobs.

# CHAPITRE 6

# JOB SHOPS AVEC OPÉRATEURS

## 6.1 INTRODUCTION

Dans le présent chapitre, nous présentons quelques extensions des résultats obtenus pour les problèmes de flow shop. Lesdits résultats ont été publiés dans [Benkalai et al. (2017a,c, 2018b)]. Nous étudions des problèmes de job shop avec un nombre d'opérateurs inférieur à celui des machines, une séquence de jobs donnée au départ et un mode de changement d'affectation libre. Par ailleurs, nous considérons également que les opérateurs ont différents niveaux de performance.

## 6.2 MINIMISATION DU MAKESPAN

Dans cette première section, nous considérons l'objectif de la minimisation du makespan dans l'environnement de job shop décrit ci-dessus.

### *6.2.1 INTRODUCTION*

Rappelons que dans le cadre de l'ordonnancement classique, il est implicitement supposé que le nombre d'opérateurs est égal au nombre de machines, ce qui permettrait à un opérateur de s'occuper exclusivement de la machine à laquelle il est affecté, et ce durant toute la durée de l'ordonnancement. Ceci est rarement vérifié dans les systèmes réels. Cette section traite le problème d'ordonnancement d'ateliers de type job shop avec un nombre d'opérateurs inférieur à celui des machines. Nous supposons que le changement d'affectation des opérateurs se fait selon le mode libre. En d'autres termes, le changement d'affectation d'un opérateur peut avoir lieu en tout temps. Nous supposons aussi que la séquence des jobs sur les machines est connue à l'avance.

Finalement, dans le but de modéliser une situation souvent rencontrée en pratique, nous supposons que les opérateurs ont des niveaux de performance différents, *i.e.* un opérateur pourrait exécuter des jobs plus rapidement ou plus lentement qu'un autre opérateur. Au meilleur de nos connaissances, ce problème n'a pas encore été traité.

### *6.2.2 DESCRIPTION DU PROBLÈME*

Le problème de job shop peut être décrit comme suit. Soit $J = \{1, 2, \ldots, n\}$ un ensemble de $n$ jobs qui doivent être exécutés par un ensemble $M = \{M_1, M_2, \ldots, M_m\}$ de $m$ machines. Chaque job $j$, $j = 1, \ldots, n$ comprend $m$ opérations $O_{ij}$ qui doivent être exécutées par la machine $M_i$, $i = 1, \ldots, m$. De plus, chaque job a sa propre "route" à travers le système, *e.g.* dans un job shop à trois machines, un job pourrait devoir passer sur les machines $M_1$, $M_3$ et $M_2$, dans cet ordre alors qu'un autre job devrait passer sur la machine $M_2$ d'abord avant de passer sur les machines $M_1$ et $M_3$.

Dans notre modèle, nous supposons qu'un job requiert la présence de l'un des $k$ opérateurs, $k < m$, durant toute la durée de son exécution. En d'autres termes, une machine restera inactive tant qu'il n'y a pas d'opérateur disponible pour la faire fonctionner. Ainsi, les temps d'exécution ne sont pas directement affectés par l'intervention des opérateurs. L'impact du partage d'opérateurs est plutôt modélisé par les temps d'attentes engendrés par le fait que le nombre d'opérateurs est inférieur à celui des machines. Les paramètres $p_{ij}$ et $C_{ij}$ représentent respectivement le temps d'exécution et le temps de fin de traitement de l'opération $(i, j)$ *i.e.* le traitement du job $j$ sur la machine $M_i$.

Nous supposons que nous avons $\ell$ types d'opérateurs, $k_h$ opérateurs de chaque type, $h = 1, \ldots, \ell$, avec $\sum_{h=1}^{\ell} k_h = k$. Un opérateur de type $\ell$ a une vitesse de travail $v_\ell$ [1]. Le changement d'affectation de ces opérateurs se fait selon le mode libre. En d'autres termes, un opérateur peut interrompre le traitement d'un job à tout moment pour traiter un autre job. Le job interrompu pourra alors être repris par le même opérateur ou par un autre. Suivant la notation présentée en Section 5.2, nous notons notre problème $J_m|res\ 1k1, S, f, v_h|C_{max}$.

L'objectif est de trouver un ordonnancement qui minimise le makespan. Dans un environnement de type job shop, nous notons la séquence de jobs $\pi = (\pi_{ij})$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Comme la séquence de jobs est donnée, l'objectif est de trouver une affectation des opérateurs $a^*$ telle que

$$C_{\max}(\pi, a^*) = \min_{a \in A} C_{\max}(\pi, a),$$

où $A$ représente l'ensemble des affectations possibles, $a^* \in A$, et $C_{max}(\pi, a)$ est le temps de complétion final de la séquence $\pi$ sous l'affectation $a$.

1. Les résultats de complexité obtenus sont identiques pour les cas où les opérateurs ont des niveaux de performance identiques.

Avant de poursuivre, décrivons la notation utilisée ici :

— Comme les opérations peuvent être interrompues, une opération $\pi(i,j)$ serait composée de $o_{ij}$ sous-opérations : $\pi(i,j)_b$, $b = 1,\ldots,o_{ij}$.

— Quand une opération n'est pas interrompue, $\pi(i,j) = \pi(i,j)_1$.

— Un opérateur $h$, $h = 1,\ldots,k$, traite $n_h$ sous-opérations : $op_c$, $c = 1,\ldots,n_h$.

— $\Delta_{ij}$ est l'ensemble des machines qui précèdent la machine $M_i$ sur la route du job $j$.

— $p_{h,op_c,\pi(i,j)_b}$ est le temps durant lequel l'opérateur $h$ traite sa $c$-ème sous-opération, $1 \leq c \leq n_h$, qui correspond à une opération $\pi(i,j)_b$, $1 \leq i \leq m$, $1 \leq j \leq n$ et $1 \leq b \leq o_{ij}$.

— $av_{h,op_c,\pi(i,j)_b}$ est l'instant auquel la $c$-ème sous-opération traitée par l'opérateur $h$ devient disponible [2].

— $ts_{h,op_c,\pi(i,j)_b}$ est l'instant auquel l'opérateur $h$ en charge de la sous-opération [3] $\pi(i,j)_b$ commence à la traiter.

— $C(\pi(i,j)_b,a)$ et $C(\pi(i,j),a)$ représentent les dates de complétion de la sous-opération $\pi(i,j)_b$ et de l'opération $\pi(i,j)$ sous l'affectation $a$. Elles sont calculées suivant les formules récursives suivantes.

Étant donnée une solution $(\pi,a)$, les dates de complétion des opérations sont calculées comme suit.

2. Cette opération corresponds à une opération $\pi(i,j)_b$, $1 \leq i \leq m$, $1 \leq j \leq n$ et $1 \leq b \leq o_{ij}$. Une sous-opération devient disponible quand toutes les sous-opérations qui la précèdent ainsi que les opérations du job $j$ sur les machines dans $\Delta_{ij}$ ont été complétées.

3. qui est sa $c$-ème sous-opération.

$$
\begin{aligned}
ts_{h,op_0,\pi(i,j)_b} &= av_{h,op_0,\pi(i,j)_b} = p_{h,op_0,\pi(i,j)_b} = -\infty;\ h = 1,\ldots,k. \\
ts_{h,op_c,\pi(i,j)_b} &= \max\left\{0, ts_{h,op_{c-1},\pi(i',j')_{b'}} + \frac{p_{h,op_{c-1},\pi(i',j')_{b'}}}{v_h}, av_{h,op_c,\pi(i,j)_b}\right\}; \\
&\quad h = 1,\ldots,k;\ c = 1,\ldots,n_h. \qquad (6.1) \\
av_{h,op_c,\pi(i,j)_b} &= \max\left\{0, \max_{i'\in\Delta_{ij}} C(\pi(i',j),a), C(\pi(i,j)_{b-1},a)\right\}; \\
&\quad h = 1,\ldots,k;\ c = 1,\ldots,n_h. \qquad (6.2) \\
C(\pi(i,j)_b,a) &= ts_{h,op_c,\pi(i,j)_b} + \frac{p_{h,op_c,\pi(i,j)_b}}{v_h}; \\
&\quad i = 1,\ldots,m;\ j = 1,\ldots,n;\ b = 1,\ldots,o_{ij}. \qquad (6.3) \\
C(\pi(i,j),a) &= C(\pi(i,j)_{o_{ij}},a);\ i = 1,\ldots,m;\ j = 1,\ldots,n. \qquad (6.4)
\end{aligned}
$$

Les équations (6.1) calculent les temps auxquels les opérateurs commencent le traitement des opérations leur étant affectées. Pour chaque opérateur, ces temps de début correspondent soit au temps où l'opération en question devient disponible ou au temps où l'opérateur termine l'opération qui la précède ou au temps 0 pour leur première opération.

Les équations (6.2) calculent les dates de disponibilité des opérations. Une opération $S(i,j)$ devient disponible lorsque toutes les opérations la précédant sur la machine $M_i$ ainsi que les opérations du job $j$ sur les machines $M_{i'}$, tel que $i' \in \Delta_{ij}$, ont été complétées.

Finalement, les équations (6.3) et (6.4) calculent les temps de complétion des opérations.

Comme pour tout problème d'optimisation, nous commençons par l'étude de la complexité. Dans ce qui suit, nous présentons les résultats de complexité pour les cas à un et deux opérateurs ainsi que quand le nombre d'opérateurs est arbitraire.

### 6.2.3 LE CAS AVEC UN NOMBRE ARBITRAIRE k D'OPÉRATEURS

**Théorème 10.** *Le problème $J_m|res\ \ell\cdot 1, S, f, v_h|C_{\max}$ est $\mathscr{NP}$-difficile quand le nombre d'opérateurs est arbitraire.*

*Démonstration.* Nous avons montré à la Section 5.4.1 que le problème était $\mathscr{NP}$-difficile pour le flow shop. Or ce dernier est un cas spécial du job shop où les jobs ont tous des routes identiques, d'où le résultat. □

Ensuite, nous développons une borne inférieure pour le problème avec $k$ opérateurs. Elle est basée sur le fait que plusieurs machines peuvent être mises en route en même temps et qu'idéalement, on voudrait avoir tous les opérateurs qui travaillent à temps plein. Cela nous a donné le résultat suivant :

$$\beta_{of}(k) = \frac{1}{k}\left[\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}\right],\ k \geq 3$$

est une borne inférieure pour le problème $J_m|res\ \ell k_h 1, S, f, v_h|C_{\max}$ pour $k \geq 1$.

### 6.2.4 LE CAS AVEC $k = 2$ OPÉRATEURS

**Théorème 11.** *Le problème $J_m|res\ 2k_h 1, S, f, v_h|C_{\max}$ est résoluble en temps polynomial.*

*Démonstration.* La complexité de ce problème est montrée via la méthode de restriction. Notons d'abord que les contraintes de précédence imposées par la séquence de jobs $S$ ainsi que par la "route" de chaque job forment un graphe de précédence. Ainsi, notre problème peut être vu comme un problème à deux machines parallèles uniformes avec des contraintes

de précédence et la possibilité de préemption, noté $Q_2|prec, prmp|C_{\max}$, où les machines, les jobs et le graphe de précédence représentent respectivement les $k = 2$ opérateurs, les $m * n$ opérations, ainsi que les contraintes de précédence entre ces opérations.

Nous savons que le problème $Q_2|prec, prmp|C_{\max}$ est résoluble en temps polynomial.

En effet,

$$Q_2|prec, prmp|C_{\max} \propto Q_2|prec, prmp, r_j|L_{\max}$$

et $Q_2|prec, prmp, r_j|L_{\max}$ est résoluble en temps polynomial [Lawler (1982)]. Il s'en suit que $J_m|res\ 2k_h 1, S, f, v_h|C_{\max}$ est aussi résoluble en temps polynomial. □

Nous présentons maintenant la méthode de résolution polynomiale induite pour notre problème par la réduction.

Soit $Succ(i, j)$ l'ensemble des successeurs de l'opération $\pi(i, j)$.

Le *niveau* $L_s(T_{\pi(i,j)})$ d'un job $T_{\pi(i,j)}$ [4] au temps $s$ est le temps minimum nécessaire pour le traitement dudit job ainsi que tous ses successeurs à partir du temps $s$. Pour notre problème, $L_s(T_{\pi(i,j)}) = \left( \sum_{(x,y) \in Succ(i,j)} p_{x,\pi(x,y)} \right) + p_{i,\pi(i,j)}$.

L'algorithme en $O(m^2 n^2)$ présenté à la Figure 6.1 est une adaptation de l'algorithme de [Horvath et al. (1977)]. Il fournit une solution avec un makespan minimum pour $k = 2$ dans le cas où les opérateurs sont identiques mais aussi quand ils ont différents niveaux de performance. En pratique, nous aurions d'abord à affecter des valeurs aux niveaux de performance des opérateurs en se basant sur la vitesse avec laquelle ils traitent les jobs. ces valeurs pourraient être évaluées par la personne en charge de l'atelier.

4. Le job à la $j^{me}$ position sur la $i^{me}$ machine dans la séquence donnée $S$.

$s \leftarrow 0$;
$h \leftarrow$ le nombre d'opérateurs libres au temps $s$;
$j \leftarrow$ le nombre de jobs de plus haut niveau au temps $s$;
**Faire**
{
**Si**($j \leq h$)
{ Les $j$ opérateurs les plus rapides traitent les $j$ jobs de façon à ce que chaque job reçoive le même temps de traitement ($^*$); }
**Sinon**
{ Les $j$ jobs sont traités pendant le même temps par les $h$ opérateurs($^*$); }
**Si**( $\exists$ opérateur libre)
{ Affecter les jobs du prochain plus haut niveau; }
**Si**(( $\exists$ $T$ qui se termine à $t$) ou ($\exists$ $T$, $T'$ / $L_s(T) > L_s(T')$ et $L_t(T) = L_t(T')$))
{ $s \leftarrow t$; }
} **Tant que**(Tous les jobs n'ont pas été traités)

Pour construire une solution avec le mode de changement d'affectation libre, les $j$ jobs qui se partagent les $h$ opérateurs durant les étapes ($^*$) reçoivent le même temps de traitement des dits opérateurs;

Chaque intervalle partagé est divisé en $j$ sous-intervalles;

Chacun des $j$ jobs est ordonnancé dans $h$ sous-intervalles, à chaque fois avec un opérateur différent;

**Figure 6.1: Algorithme pour la résolution du problème $J_m|res\ 2k_h1, S, v_h|C_{max}$.**

### *6.2.5 LE CAS AVEC $k = 1$ OPÉRATEUR*

Rappelons que le cas à un opérateur est un cas particulier de celui à deux opérateurs. Dans ce qui suit, nous décrivons un algorithme plus simple pour sa résolution.

**Théorème 12.** *Le problème $J_m|res\ 111, S|C_{\max}$ est résoluble en temps polynomial.*

*Démonstration.* Nous utilisons la même technique de réduction que celle utilisée pour le cas avec $k = 2$. Nous utilisons pour la réduction le problème $1|prec, prmp|C_{\max}$ qui est résoluble

en temps polynomial. En effet,

$$1|prec, prmp|C_{max} \propto 1|prec, prmp, r_j|L_{max}$$

et $1|prec, prmp, r_j|L_{\max}$ est résoluble en temps polynomial [Baker et al. (1983)]. □

Comme précédemment mentionnée, le concept Un opérateur-Plusieurs machines est populaire et souvent rencontré en pratique, particulièrement dans les systèmes de production juste-à-temps.

Dans le cas avec un seul opérateur, la somme de tous les temps de traitement $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$ est une borne inférieure évidente. Nous remarquons que toute solution *sans retard*[5] a un makespan égal à cette borne inférieure. Son optimalité est ainsi établie. Une telle solution peut être construite en $O(mn)$.

Une politique d'ordonnancement optimale pourrait être de traiter, sans interruption, toutes les opérations du job 1, ensuite toutes celles du job 2 et ainsi de suite jusqu'au job $n$.

### *6.2.6 CONCLUSION*

Dans cette section, nous avons étudié l'affectation d'un nombre d'opérateurs inférieur au nombre de machine dans un atelier de type job shop avec une séquence de jobs donnée au départ et le mode de changement d'affectation libre pour la minimisation du makespan. Les opérateurs considérés ont différents niveaux de performance. Nous avons étudié la complexité de notre problème dans le cas avec un et deux opérateurs ainsi que lorsque leur nombre est arbitraire. Dans ce dernier cas, le problème est $\mathscr{NP}$-difficile, même lorsque les opérateurs ont

5. Une solution dans laquelle aucune machine ne reste inactive tant qu'elle a une opération disponible pour traitement ainsi qu'un opérateur disponible pour la faire fonctionner.

des niveaux de performance équivalents. Nous avons par la suite présenté une borne inférieure qui pourrait être utilisée pour l'évaluation de la qualité de futures méthodes de résolution pour ce cas. Pour les cas avec un et deux opérateurs, nous avons présenté des méthodes de résolution polynomiales.

## 6.3 MINIMISATION DU RETARD ALGÉBRIQUE MAXIMUM

Dans la présente section, nous considérons l'objectif du retard algébrique maximum.

### *6.3.1 INTRODUCTION*

Dans la présente section, nous étudions l'ordonnancement d'ateliers de type job shop en présence d'opérateurs dont l'affectation change selon un mode libre, où l'ordre de tâches est connu au départ et l'objectif est la minimisation du retard algébrique maximum. Les opérateurs ont des niveaux différents de performance. À notre connaissance, ce problème n'a pas encore été étudié.

### *6.3.2 DESCRIPTION DU PROBLÈME*

Dans un problème de job shop, $n$ tâches doivent être traitées par $m$ machines. Chaque tâche $j$, $j = 1,\ldots,n$ doit être traitée par chacune des machines $i$, $i = 1,\ldots,m$ et ce dans un ordre fixe qui lui est propre.

Dans notre modèle, nous supposons qu'une opération nécessite la présence de l'un des $k$ opérateurs ($k < m$) pour toute la durée de son traitement. L'impact de la présence d'un nombre d'opérateurs inférieur au nombre de machines y est ainsi modélisé par des temps d'attente

durant lesquels une machine restera inactive même si elle a des opérations en attente. Les paramètres $p_{ij}$, $d_{ij}$ et $C_{ij}$ représentent respectivement le temps de traitement, la date d'échéance et la date de fin de traitement de la tâche $j$ sur la $i^{\text{ème}}$ machine.

Nous avons $\ell$ types d'opérateurs, $k_h$, $h = 1, \ldots, \ell$, opérateurs de chaque type avec $\sum_{h=1}^{\ell} k_h = k$. Un opérateur de type $h$ a une cadence $v_h$. Selon le mode libre, le changement d'affectation des opérateurs peut se produire à tout instant. Un opérateur pourra alors interrompre le traitement d'une tâche pour s'occuper d'une autre tâche. Cette tâche pourra par la suite être reprise par ce même opérateur ou être confiée à un autre opérateur. L'objectif est de minimiser le retard algébrique maximum, $L_{\max} = \max_{j=1,\ldots,n} L_{mj}$ où $L_{ij} = C_{ij} - d_{ij}$. Les dates de fin de traitement sont calculées selon les équations récursives présentées à la Section 6.2.2. Dans un environnement de type job shop, nous notons la séquence de jobs $\pi = (\pi(i,j))$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, où $\pi(i,j)$ représente l'opération du job $j$ sur la machine $M_i$. L'objectif est de trouver une affectation des opérateurs $a^*$ telle que

$$L_{\max}(\pi, a^*) = \min_{a \in A} L_{\max}(\pi, a),$$

où $A$ représentes l'ensemble des affectations possibles, $a^* \in A$, et $L_{max}(\pi, a)$ est le retard algébrique maximum des jobs de la séquence $\pi$ sous l'affectation $a$.

Dans ce qui suit, nous étudions la complexité de notre problème pour les cas à 1, 2 et plus de 3 opérateurs. Pour les deux premiers cas, nous proposons des méthodes de résolution de complexité polynomiale.

Nous notons notre problème $J_m|res\ \ell k_h 1, S, v_h, f|L_{\max}$ conformément à la notation décrite à la Section 5.2. Rappelons que nous ajoutons les symboles $S$ pour indiquer que l'affectation se fait sous un ordre de tâches donné $S$, $v_h$ qui indique que les opérateurs ont des cadences

différentes et $f$ qui indique que le mode de changement d'affectation est libre.

### *6.3.3 LE CAS À k OPÉRATEURS, k ARBITRAIRE*

**Théorème 13.** *Le problème $J_m|res\ \ell k_h 1, S, v_h, f|L_{\max}$ est $\mathscr{NP}$-difficile quand k est arbitraire.*

*Démonstration.* Nous avons montré à la Section 5.4.2 que le problème était $\mathscr{NP}$-difficile pour le problème du flow shop qui se trouve être un cas spécial du job shop où les jobs ont tous des routes identiques, d'où le résultat. □

### *6.3.4 LE CAS À $k = 2$ OPÉRATEURS*

**Théorème 14.** *Le problème $J_m|res\ 2k_h 1, S, v_h, f|L_{\max}$ est résoluble en temps polynomial.*

*Démonstration.* La preuve se fera via la méthode de restriction. Les contraintes de passage des opérations sur les machines et celles générées par $S$, forment un graphe de précédence. Ainsi, notre problème peut être vu comme un problème de deux machines parallèles uniformes avec un graphe de précédence et la possibilité de préemption, noté par $Q_2|prec, prmp|L_{\max}$, où les machines, les tâches et le graphe de précédence représentent respectivement les 2 opérateurs, les $m * n$ opérations, et les relations de précédence entre elles. La possibilité de préemption modélise le mode de changement d'affectation libre. Ce problème est résoluble en temps polynomial [Lawler (1982)]. □

Passons à la méthode de résolution. Les algorithmes suivants sont des adaptations de ceux décrits par Lawler dans [Lawler (1982)].

La solution est construite en trois étapes. Il s'agit de modifier d'abord les dates d'échéance

(Figure 6.2) puis de construire des intervalles d'ordonnancement variables (Figure 6.3). Enfin, une solution est obtenue avec un algorithme d'ordonnancement par priorité (Figure 6.4) en $O(m^2n^2)$.

Soient les données suivantes :

- $d_{ij} = d_j - \sum_{q=i+1}^{m} p_{qj}$ la date d'échéance de la $i^{me}$ opération de la tâche $j$.
- Sans perdre de généralité, $v_1 = 1$, $v_2 = \frac{min\{v_1,v_2\}}{max\{v_1,v_2\}}$.
- $p_{ij}(t)$ le temps de traitement minimum qui doit être effectué sur l'opération $(i, j)$ avant la date $t$ pour que ladite opération respecte sa date d'échéance.
- $S(i, j)$ l'ensemble des successeurs (immédiats ou pas) de l'opération $(i, j)$.

Le paramètre $b_{ij}^{(r)} = d_{ij} - p_{ij}^{(r)}$, où $p_{ij}^{(r)}$ est le temps de traitement restant à effectuer sur l'opération $(i, j)$ à l'instant $t_r$ ; il sert d'indicateur de priorité. Plus $b_{ij}^{(r)}$ est petit, plus la priorité de l'opération $(i, j)$ est grande. Les $z$ opérations disponibles à un instant $t_r$ seront ré-indexées selon l'ordre croissant des $b_{ij}^{(r)}$.

Soit $x_{ij}^{(r)}$ la quantité de l'opération $(i, j)$ exécutée durant l'intervalle $[t_r, t_{r+1}]$.

$$x_{ij}^{(r)} = \begin{cases} \Delta, & q < u, \\ max\{0, T - b_q^{(r)}\}, & q \geq u, \end{cases}$$

avec $T$, $u$, $v$ et $\Delta$ déterminés par l'algorithme en Figure 6.3, $v$ étant l'indice tel que $b_v^{(r)} \leq T < b_{v+1}^{(r)}$ et $q$ l'indice correspondant à l'opération $(i, j)$ après ré-indexation.

**Pour** (toute opération $(i,j)$ sans successeur)
$\{d'_{ij} \leftarrow d_{ij}\,;\}$
Créer deux listes ordonnées par date d'échéance modifiées $d'_{ij}$ et $b'_{ij} = d'_{ij} - p_{ij}$ ;
**Tant que** ($\exists\ (i,j)$ / $d_{ij}$ non modifiée et $d_{i'j'}$ modifiée $\forall (i',j') \in S(i,j)$)
{ Choisir $(i,j)$ ;
Chercher les successeurs de cette opération dans les deux listes ordonnées ;
Calculer $\sum_{(i',j')\in S(i,j)} p_{i'j'}(d'_{i''j''})\ \forall (i'',j'') \in S(i,j)$ ;

$$d'_{ij} \leftarrow min\left\{ d_{ij}, \min_{d'_{(i'',j'')}/(i'',j'')\in S(i,j)} \left\{ d'_{(i'',j'')} - \frac{\sum_{(i',j')\in S(i,j)} p_{i'j'}(d'_{i''j''})}{v_1+v_2} \right\} \right\} ;$$

**Pour** (chaque $(i',j') \in S(i,j)$)
$\{d'_{ij} \leftarrow min\{d'_{ij}, b'_{i'j'} = d'_{i'j'} - p_{i'j'}\}\,;\}$
Insérer $d'_{ij}$ et $b'_{ij} = d'_{ij} - p_{ij}$ dans les listes ordonnées ;
}

**Figure 6.2: Algorithme pour la modification des dates d'échéance.**

$\Delta \leftarrow 0\,;\ u \leftarrow 2\,;\ v \leftarrow 2\,;\ T \leftarrow b_2\,;\ P \leftarrow 0\,;$
**Tant que** $(\Delta < t_{r+1} - t_r)$
{
**Tant que** $(T = b_{u-1} + \Delta)$
$\{\ u \leftarrow u-1$ (Si $u = 1$, $b_0 = -\infty)\,;\ P \leftarrow P + \Delta\,;\}$
**Tant que** $(T = b_{v+1})$
$\{\ v \leftarrow v+1$ (Si $v = z$, $b_{z+1} = \infty)\,;\}$
$T_1 \leftarrow (\frac{(b_{u-1}+\Delta)(2-u+v_2)-(v-u+1)T)}{((2-u+v_2)-(v-u+1))}\,;\ T_2 \leftarrow b_{v+1}\,;\ T_3 \leftarrow T + \frac{(t_{r+1}-t_r-\Delta)(2-u+v_2)}{(v-u+1)}\,;$
$T' \leftarrow min\{T_1, T_2, T_3\}\,;\ P \leftarrow P + (v-u+1)(T'-T)\,;\ \Delta \leftarrow \frac{P}{(2-u+v_2)}\,;\ T \leftarrow T'\,;$
}

**Figure 6.3: Algorithme pour la construction d'intervalles fixes.**

### *6.3.5 LE CAS À $k = 1$ OPÉRATEUR*

Ici encore, rappelons que le cas à un opérateur est un cas particulier de celui à deux opérateurs.

Dans ce qui suit, nous décrivons un algorithme plus simple pour sa résolution.

**Théorème 15.** *Le problème $J_m|res\ 111, S, f|L_{\max}$ est résoluble en temps polynomial.*

Calculer le degré intérieur de chaque opération $(i, j)$ ;
Créer une file $Q$ contenant les opérations de degré intérieur égal à 0 ;
Créer une liste $A$ contenant les opérations disponibles pour traitement* ;
$r \leftarrow 0$ ;
**Tant que** $(Q \neq \varnothing)$
{ $r \leftarrow r+1$ ; $t_r \leftarrow 0$ ; $A \leftarrow A \cup Q$ ; $Q \leftarrow \varnothing$ ;
**Tant que** $(A \neq \varnothing)$
{ $t_{r+1} \leftarrow 0$ ;
Exécuter l'algorithme en Figure 6.3 sur les opérations dans $A$ ;
**Si** (les valeurs trouvées $(\Delta, u, v, T)$ violent $b_j + \Delta \leq d_j$, $j = 1, \ldots, u-1$)
{ Ré-exécuter l'algorithme en Figure 6.3 avec $t_{r+1} = t_r + \min_y \{p_j^y / j = 1, \ldots, u-1\}$ ;
$A \leftarrow A \backslash \{$opérations complétées dans$[t_r, t_{r+1}]\}$ ;
Mettre à jour les degrés intérieurs des opérations ;
Mettre à jour $Q$ ; $A \leftarrow A \cup Q$ ; $k \leftarrow k+1$ ;
}
}
}

N.B : Pour la construction d'intervalles variables, on procède de manière analogue à la différence des points suivants :

— $T \leq d_j$, $j = u, u+1, \ldots, v$. Si $T = d_j$, $j \geq u$, le calcul s'arrête.

— Si $u$ est décrémenté, on vérifie si $b_{u-1} + \Delta > d_{u-1}$, si c'est le cas, le calcul s'arrête.

* Une opération est dite disponible à la date $t$ si le traitement de tous ses prédécesseurs est complété et s'il reste un temps de traitement non nul à effectuer sur cette opération.

**Figure 6.4: Algorithme pour la résolution de $J_m|res\ 2k_h 1, S, v_h, f|L_{\max}$.**

*Démonstration.* Nous utilisons le même principe de preuve que précédemment. Nous utilisons pour la réduction $1|prec, prmp, r_j|L_{\max}$ qui est résoluble en temps polynomial [Baker et al. (1983)]. □

L'algorithme présenté en Figure 6.5 est une adaptation de l'algorithme de [Baker et al. (1983)] en $O(m^2n^2)$ pour le problème $1|prec, prmp, r_j|L_{max}$. Dans notre cas, toutes les dates de disponibilités $(r_j)$ sont égales à 0 et ainsi toutes les opérations appartiennent à un même block $B$ et peuvent être traitées sans temps d'arrêt. Soit $P(B) = \sum_{(i,j) \in B} p_{ij}$.

**Tant que** $(B \neq \varnothing)$
{ Choisir $(i,j)$ / $S(i,j) = \varnothing$ et $L_{ij} = \min_{(i,j)\in B} \{P(B) - d_{ij}\}$ ;
$B \leftarrow B \backslash \{(i,j)\}$ ;
}

**Figure 6.5: Algorithme pour la résolution de** $J_m|res\ 111, S, f|L_{\max}$**.**

### *6.3.6 CONCLUSION*

Dans la présente section, nous avons étudié le problème d'affectation d'opérateurs ayant différents niveaux de performances dans un environnement de type job shop. Le nombre d'opérateurs est inférieur à celui des machines, l'ordre des tâches est donné, le mode de changement d'affectation est libre et l'objectif est la minimisation du retard algébrique maximum. Ce problème est fortement NP-difficile pour $k \geq 3$. Des algorithmes de complexité polynomiale ont été fournis pour les cas à un et deux opérateurs. Il serait intéressant de développer des heuristiques pour les cas à trois opérateurs et plus, mais aussi de considérer d'autres objectifs.

# CHAPITRE 7

# OPEN SHOPS AVEC OPÉRATEURS

## 7.1 INTRODUCTION

Dans le présent chapitre, nous présentons quelques extensions des résultats obtenus pour les problèmes de flow shop. Lesdits résultats ont été publiés dans [Benkalai et al. (2018a)]. Nous étudions des problèmes d'open avec un nombre d'opérateurs inférieur à celui des machines et un mode de changement d'affectation en fin de tâche.

Par ailleurs, nous étudions deux variantes du problème. Premièrement, le cas où la séquence de tâches est fixée pour chaque machine et où on affecte les opérateurs par la suite. Deuxièmement, le cas où l'on gère simultanément l'affectation d'opérateurs et l'ordonnancement de tâches. Nous considérons la minimisation de deux objectifs : d'abord, le temps de traitement total aussi appelé makespan ; ensuite, le retard algébrique maximum.

## 7.2 DESCRIPTION DU PROBLÈME

Le problème d'open shop classique peut être formulé comme suit. Nous avons un ensemble $J = \{1, 2, \ldots, n\}$ de $n$ tâches et un ensemble $M = \{M_1, M_2, \ldots, M_m\}$ de $m$ machines. Chaque

tâche $j$ doit être exécutée sans préemption par chaque machine $M_i$, $1 \leq i \leq m$ ; ainsi, on peut voir une tâche j comme un ensemble de $m$ opérations, $O_{1j}, \ldots, O_{mj}$. À chaque opération $O_{ij}$ on associe les paramètres $p_{ij}$, $d_{ij}$ et $C_{ij}$ qui représentent respectivement le temps de traitement, la date d'échéance et la date de fin de traitement de la tâche $j$ sur la machine $M_i$. Dans le cadre de l'open shop, la "route" de chaque tâche à travers le système n'est pas prédéfinie. Dans notre modèle, nous supposons que chaque opération a besoin d'un opérateur pour toute la durée de son exécution. En d'autres termes, une machine ayant des opérations en attente restera inactive tant qu'un opérateur ne sera pas disponible pour effectuer le traitement requis. Les opérateurs sont au nombre de $k$, avec $k < m$. Sous les précédentes hypothèses, le partage d'opérateurs n'affectera pas directement les temps de traitement des tâches. Au lieu de cela, ce sont les retards engendrés par le manque d'opérateurs qui modéliseront cet impact sur le système.

Au cours de l'exécution de l'ensemble de tâches, nous aurons à changer l'affectation des opérateurs. Ce changement se fait selon le mode "fin d'opération" où un opérateur n'est pas autorisé à interrompre l'opération qu'il traite avant la fin de celle-ci.

Les objectifs à minimiser sont les suivants :

1. Le temps de traitement total, $C_{\max} = \max_{j=1,\ldots,n} \{ \max_{i=1,\ldots m} \{C_{ij}\}\}$.
2. Le retard algébrique maximum, $L_{\max} = \max_{j=1,\ldots,n} \{ \max_{i=1,\ldots m} \{L_{ij}\}\}$ où $L_{ij} = C_{ij} - d_{ij}$.

Pour décrire nos problèmes, nous avons utilisé la notation présentée à la Section 5.2.

## 7.3 ÉTUDE DE COMPLEXITÉ

Conformément à la notation décrite en Section 5.2, nous notons nos problèmes comme suit :

1. Problème d'affectation d'opérateurs sur une séquence donnée pour la minimisation du

makespan $O_m|res\ 1k1, S, e|C_{\max}$ et pour la minimisation du retard algébrique maximum $O_m|res\ 1k1, S, e|L_{\max}$

2. Problème de gestion simultanée des opérateurs et des tâches pour la minimisation du makespan $O_m|res\ 1k1, e|C_{\max}$ et pour la minimisation du retard algébrique maximum $O_m|res\ 1k1, e|L_{\max}$

On commence notre étude de complexité par le cas trivial d'un opérateur.

### *7.3.1 CAS DE $k = 1$ OPÉRATEUR*

On montre que les problèmes avec un opérateur sont résolubles en temps polynomial quand la séquence est fixée mais aussi quand elle ne l'est pas, et ce, pour les deux objectifs.

**Lemme 11.** $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$ *est une borne inférieure pour les problèmes* $O_m|res\ 111, S, e|C_{\max}$ *et* $O_m|res\ 111, e|C_{\max}$.

*Démonstration.* : Évidente. □

**Théorème 16.** *Les problèmes* $O_m|res\ 111, S, e|C_{\max}$ *et* $O_m|res\ 111, e|C_{\max}$ *sont résolubles en temps polynomial.*

*Démonstration.* On voit clairement qu'un algorithme qui fait que l'opérateur traite toutes les opérations de la machine $M_1$, puis $M_2$ et ainsi de suite jusqu'à la machine $M_m$ sans temps de pause fournit une solution avec un makespan égale à la borne inférieure du Lemme 11. Cet algorithme a un temps d'exécution en $O(mn)$ ce qui fait que les problèmes sont résolubles en temps polynomial. □

**Théorème 17.** *Le problème* $O_m|res\ 111, e|L_{\max}$ *est résoluble en temps polynomial.*

*Démonstration.* Nous voyons bien que notre problème est équivalent à un problème à machine unique pour la minimisation du retard algébrique maximum. D'après [Pinedo (2002)], ce dernier est résoluble en temps polynomial, ce qui fait que le notre l'est aussi. □

**Théorème 18.** *Le problème* $O_m|res\ 111, S, e|L_{\max}$ *est résoluble en temps polynomial.*

*Démonstration.* Nous voyons bien que notre problème est équivalent à un problème à machine unique pour la minimisation du retard algébrique maximum avec des contraintes de précédence sous forme de chaines. D'après [Pinedo (2002)], $1|prec|L_{max}$ est résoluble en temps polynomial pour des contraintes de précédence quelconques, ce qui fait que notre problème est aussi résoluble en temps polynomial. □

Maintenant, nous nous intéressons aux problèmes avec $k \geq 2$ opérateurs, nous nous inspirons d'une partie du papier de [Benkalai et al. (2017b)].

### *7.3.2 AFFECTATION D'OPÉRATEURS ÉTANT DONNÉE UNE SÉQUENCE DE TÂCHES*

Ici, nous voulons prouver que le problème d'affectation d'opérateurs selon une séquence de tâches est $\mathscr{NP}$-difficile. Pour ce faire, nous nous ramenons au problème de décision suivant :

$OSMS(k,m)$ est un problème d'open shop avec $\ell$ tâches, $m \geq 3$ machines et $k \geq 2$ opérateurs qui changent d'affectation selon le mode en fin d'opération et $S$ est une séquence de tâches sur les machines ; existe-t-il une affectation $a \in A$ telle que $C_{max}(S,a) \leq \alpha$ ?

Nous montrons dans ce qui suit que le problème $OSMS(2,3)$ est $\mathscr{NP}$-complet. La réduction est construite à partir du problème de décision à deux machines parallèles :

*PM* est un problème à deux machines parallèles avec $n$ tâches et une structure de contraintes de précédence sous forme de $\ell$ chaînes, où $p_h$ est le temps de traitement de la tâche $h$, $1 \leq h \leq n$.

Étant donné un entier $\alpha$, existe-t-il un ordonnancement $\pi$ tel que $C_{max}(\pi) \leq \alpha$ ?

Soit $I$ une instance du problème $PM$. On construit une instance $I'$ de $OSMS(2,3)$ comme suit. On note $C_w$, $w = 1, \ldots, \ell$, les chaines de l'instance $I$. L'ensemble de tâches de $I'$ contient $\ell$ tâches où chaque tâche a exactement une opération sur l'une des trois machines :

— si $w \equiv 0 \mod 3$, alors une tâche $w$ est associée à la machine $M_1$ avec un temps de traitement égal à la somme des temps de traitement des tâches dans $C_w$ ; *i.e.* $p_{1w} = \sum_{h \in C_w} p_h$, $p_{2w} = p_{3w} = 0$.

— si $w \equiv 1 \mod 3$, alors une tâche $w$ est associée à la machine $M_2$ avec un temps de traitement égal à la somme des temps de traitement des tâches dans $C_w$ ; *i.e.* $p_{2w} = \sum_{h \in C_w} p_h$ ; $p_{1w} = p_{3w} = 0$.

— si $w \equiv 2 \mod 3$, alors une tâche $w$ est associée à la machine $M_3$ avec un temps de traitement égal à la somme des temps de traitement des tâches dans $C_w$ ; *i.e.* $p_{3w} = \sum_{h \in C_w} p_h$ ; $p_{1w} = p_{2w} = 0$.

On aura une contrainte de précédence entre les tâches $w$ et $w+3$, $w = 1, \ldots, \ell - 3$. Ceci correspondra à la séquence de tâches de $I'$. Pour finir, on pose $\alpha = \lceil \ell/2 \rceil \max_{1 \leq w \leq \ell} \{\text{Length}(C_w)\}$, où $\text{Length}(C_w)$ est la somme des temps de traitement des tâches dans $C_w$.

Nous voyons bien que cette construction peut se faire en temps polynomial.

Maintenant, on montre que l'instance $I$ a la réponse oui si, et seulement si, $I'$ a aussi une réponse oui.

**Lemme 12.** *Si I a la réponse oui alors il y a au moins une solution où $\lceil \ell/2 \rceil$ chaînes sont traitées par la machine 1 et le restant d'entre elles par la machine 2.*

*Démonstration.* Comme les chaînes sont indépendantes, on peut en traiter $\lceil \ell/2 \rceil$ l'une après l'autre sur la machine 1 sans dépasser $\alpha$. Les chaînes restantes seront traitées par la machine 2

toujours sans dépasser $\alpha$. □

**Lemme 13.** *Si $I$ a la réponse oui, alors $I'$ a la réponse oui.*

*Démonstration.* Si $I$ a la réponse oui, nous avons un ordonnancement $\pi$ tel que $C_{max}(\pi) \leq \alpha$.

Sans perdre de généralités, on suppose que $\pi$ a la structure décrite dans le Lemma 12.

À partir de cette solution, on peut construire une solution à $I'$ où l'opérateur 1 (2) traite les tâches de la machine $M_1$ ($M_2$) de $PM$. Comme les ensembles de tâches sur chaque machine du flow shop sont indépendants des autres, un tel ordonnancement est valide sans dépasser $\alpha$. Ainsi, $I'$ a la réponse oui. □

**Lemme 14.** *Si $I'$ a la réponse oui, alors il existe au moins une solution telle qu'un opérateur traite $\lceil \ell/2 \rceil$ tâches alors que l'opérateur traite les tâches restantes.*

*Démonstration.* Si un opérateur traite $\lceil \ell/2 \rceil$ tâches, son temps de travail ne dépasse pas $\alpha$. Ce sera aussi le cas pour l'autre opérateur.

□

**Lemme 15.** *Si $I'$ a la réponse oui, alors $I$ a la réponse oui.*

*Démonstration.* Si $I'$ a la réponse oui, alors nous avons une affectation $a$ et une séquence $S$ telles que $C_{max}(S,a) \leq \alpha$.

Les traitement des $\ell$ tâches de $I'$ avec deux opérateurs est équivalent au traitement des $\ell$ chaînes de $I$ sur les deux machines de $PM$. Dans $PM$, les chaînes traitées le seront sans interruption, ce qui induit un temps total de traitement $C_{max}(\pi) \leq \alpha$. Ainsi, $I$ a la réponse oui. □

**Théorème 19.** *Le problème $O_m|res\ 1k1, S, e|C_{max}$ $\mathscr{N}\mathscr{P}$-difficile au sens fort pour $k \geq 2$.*

*Démonstration.* Le problème $OSMS(2,3)$ est dans $\mathscr{N}\mathscr{P}$. Par les Lemmes 13 et 15, nous avons une équivalence de l'existence de solutions pour les deux problèmes. Le résultat du théorème est ainsi établi. □

**Corollaire 3.** *Le problème $O_m|res\ 1k1,S,e|L_{max}$ $\mathscr{N}\mathscr{P}$-difficile au sens fort pour $k \geq 2$.*

*Démonstration.* Par réduction évidente de $C_{max}$ à $L_{max}$. □

### *7.3.3 GESTION SIMULTANÉE DES OPÉRATEURS ET DES TÂCHES*

Ici, nous voulons démontrer la $\mathscr{N}\mathscr{P}$-difficulté du problème de gestion simultanée des opérateurs et des tâches. La preuve est basée sur les travaux [Agnetis et al. (2011); Benkalai et al. (2017b)].

On cherche à montrer que le problème de décision suivant est $\mathscr{N}\mathscr{P}$-complet pour $k \geq 2$ :

$OSM(k)$ est un problème d'open shop avec $n$ tâches, $m = 3$ machines et $k \geq 2$ opérateurs dont le changement d'affectation se fait selon le mode fin d'opération.

Étant donné un entier $\alpha$, existe-t-il un ordonnancement $S$ et une affectation $a \in A$ tels que $C_{max}(S,a) \leq \alpha$ ?

Pour ce faire, nous prouvons que $OSM(2)$ est $\mathscr{N}\mathscr{P}$-complet. Pour la réduction, nous avons choisi le problème de partition [Garey et Johnson (1979)] que l'on défini comme suit :

Étant donné un ensemble $A = \{a_1, a_2, \ldots, a_n\}$ de cardinalité $n$ et des tailles $s(a_j) \in \mathbb{Z}^+$, $j = 1,2,\ldots,n$, existe-t-il un sous-ensemble $A' \subseteq A$ tel que $\sum_{a_j \in A'} s(a_j) = \sum_{a_j \in A - A'} s(a_j)$ ?

Soit $I$ une instance du problème de partition, on construit une instance $I'$ de $OSM(2)$ comme suit. L'ensemble de tâches contient $n+2$ tâches ayant une seule opération sur l'une des trois

machines.

- $B = \sum_{j=1}^{n} s(a_j)$,
- $p_{1j} = p_{2j} = 0, p_{3j} = s(a_j), j = 1, \ldots, n$,
- $p_{1,n+1} = p_{2,n+2} = L$,
- $p_{2,n+1} = p_{3,n+1} = p_{1,n+2} = p_{3,n+2} = 0$,
- $\alpha = L + (B/2)$, où $L > (B/2), L \in \mathbb{Z}^+$.

Une telle construction se fait clairement en temps polynomial.

Dans ce qui suit, on montre que $I$ a la réponse oui si et seulement si $I'$ a la réponse oui.

**Lemme 16.** *Si I a la réponse oui, alors I′ a la réponse oui.*

*Démonstration.* Si $I$ a la réponse oui, alors il existe $A' \subseteq A$ tel que $\sum_{a_j \in A'} s(a_j) = \sum_{a_j \in A - A'} s(a_j) = B/2$.

Ceci induit que les tâches de la troisième machine peuvent être divisées en deux ensembles dont chacun aura un temps d'exécution total de $B/2$ sans causer de préemptions. Ensuite, l'opérateur 1 traitera la tâche $n+1$ et les tâches de $M_3$ associées à $A'$, tandis que l'opérateur 2 traitera sur $M_3$ les tâches associées à $A - A'$, ensuite la tâche $n+2$, tel qu'illustré dans la Figuree 7.1. Cette solution a clairement $C_{max} = L + B/2 \leq \alpha$. Ainsi, $I'$ a la réponse oui. □

**Lemme 17.** *Si I′ a la réponse oui, alors dans sa solution, chaque opérateur traite une opération de longueur L et des opérations sur $M_3$ dont la somme des temps de traitement est $B/2$.*

*Démonstration.* Tout d'abord, on voit que chaque opérateur traite exactement une tâche de longueur $L$. En effet, tout autre configuration induirait soit une solution non valide ou alors un $C_{max}$ dépassant $\alpha$.

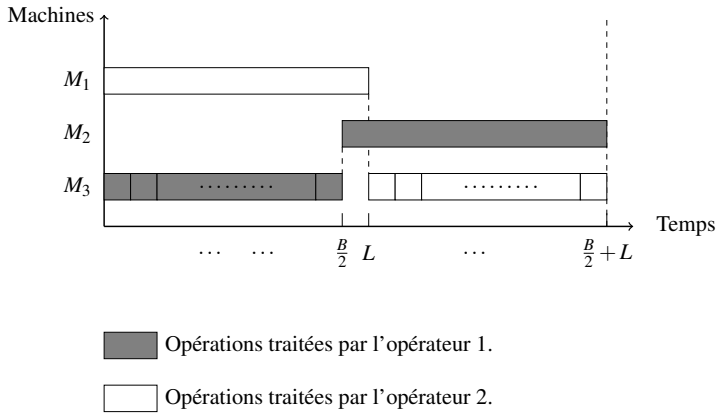



**Figure 7.1: Une solution pour $I'$ du problème $OSM(2)$.**

Ensuite, pour ne pas dépasser $\alpha$, chaque opérateur traitera un ensemble de tâches sur $M_3$ dont le temps de traitement total est $B/2$. Ainsi, une solution de $I'$ aura $C_{max}(S,a) = L + (B/2)$ et les deux opérateurs n'auront aucun temps de pause. Une solution $(S,a)$ à $I'$ est illustrée en Figure 7.1. □

**Lemme 18.** *Si $I'$ a la réponse oui, alors $I$ a la réponse oui.*

*Démonstration.* Si $I'$ a la réponse oui, alors par le Lemme 17, on sait que chaque opérateur traite des opérations sur la machine $M_3$ dont le temps de traitement total est de $B/2$. Comme les préemptions ne sont pas autorisée, on en conclut qu'il existe $A' \subseteq A$ tel que $\sum_{a_j \in A'} s(a_j) = \sum_{a_j \in A - A'} s(a_j) = B/2$. Ainsi, $I$ a la réponse oui. □

**Théorème 20.** *Le problème $O_m|res\ 1k1,e|C_{max}$ est $\mathscr{NP}$-difficile au sens faible pour $k \geq 2$.*

*Démonstration.* Le problème $OSM(2)$ est dans $\mathscr{NP}$. La preuve du théorème découle des Lemmes 16 et 18. □

**Théorème 21.** *Le problème $O_m|res\ 1k1,e|L_{max}$ est $\mathscr{NP}$-difficile au sens faible pour $k \geq 2$.*

*Démonstration.* Par réduction évidente de $C_{max}$ à $L_{max}$. □

## 7.4 CONCLUSION

Dans la présente section, nous avons étudié la complexité de problèmes d'ateliers de type open shop avec opérateurs. Nous avons étudié le cas où l'affectation des opérateurs se faisait selon une séquence de tâches donnée et aussi pour le cas où la gestion des opérateurs et des tâches se fait de manière simultanée. L'étude s'est faite pour deux objectifs, le temps de traitement total et le retard algébrique maximum.

Le nombre d'opérateurs est inférieur au nombre de machines et le changement d'affectation des opérateurs se fait selon le mode fin d'opération.

Pour toutes les configurations étudiées, nous avons trouvé que le problème avec un seul opérateur était résoluble en temps polynomial alors que les problèmes étaient $\mathscr{NP}$-difficiles pour un nombre d'opérateurs $k \geq 2$.

Ceci nous pousse à orienter le développement de futurs algorithmes vers des méthodes approchées, donc des heuristiques, métaheuristiques ou encore des matheuristiques.

Notons que la $\mathscr{NP}$-complétude au sens faible du problème de gestion simultanée des opérateurs et des tâches indique la possibilité de conception d'un algorithme pseudo-polynomial.

Par ailleurs, il serait aussi intéressant de considérer d'autres objectifs comme la somme des temps de fin de traitement ou encore la somme des pénalités.

# CONCLUSION GÉNÉRALE

L'importance théorique et pratique des problèmes d'ateliers leur ont valu un grand intérêt de la part de la communauté scientifique. Dans le contexte économique actuel, qui est fortement concurrentiel, il devient de plus en plus important pour les compagnies d'avoir des systèmes de décision adéquats et une gestion efficace de leurs ressources dans le but de fournir des services de haut niveau de performance. Dans ce contexte, il est nécessaire de réduire l'écart entre la recherche académique et les applications réelles en développant des modèles plus réalistes.

Ainsi, la prise en compte des ressources humaines dans les systèmes de production est un domaine qui vaut la peine d'être étudié. Elle permettrait d'améliorer la flexibilité des systèmes sus-mentionnés tout en éliminant des dépenses non justifiées. Ces nouveaux problèmes remettent en cause l'ordonnancement classique et présentent de nouvelles contraintes (*e.g.* règles de régulation du travail) et de nouveaux objectifs (*e.g.* les coûts). L'intégration des ressources humaines dans les systèmes de production est un domaine de recherche relativement récent, peu étudié et qui reste largement ouvert.

Son objectif est principalement le développement d'approches intégrées se basant sur une modélisation efficace des opérateurs humains dans leur milieu de travail, particulièrement

l'aspect lié à l'impact du partage d'opérateurs sur les temps d'exécution des jobs qu'ils traitent.

Dans le cadre de cette thèse, nous avons étudié l'ordonnancement d'ateliers avec contraintes d'opérateurs. Tout d'abord, dans le cas du flow shop de permutation avec temps de réglages, nous avons adapté puis amélioré la métaheuristique Migrating Birds Optimization. Pour ce problème, nous avons supposé que le nombre d'opérateurs est égal au nombre de machines et que ces derniers s'occupent des réglages. Les méthodes développées nous ont permis d'obtenir des solutions de bonne qualité et structurellement variées, ce qui ajoute de la flexibilité lors de la prise de décision.

Par la suite, nous avons considéré un cas plus présent en pratique qui est celui où le nombre d'opérateurs est inférieur à celui des machines et où les opérateurs sont donc la ressource "goulot". Nous avons considéré deux modes de changement d'affectation des opérateurs, à savoir, le mode en fin d'opération où un opérateur ne peut interrompre une opération en cours et ne peut changer d'affectation qu'à la fin de celle-ci et le mode libre où un opérateur peut interrompre une opération en tout temps. Dans ce cadre, nous avons étudié deux approches de résolution dans le but d'évaluer l'importance des deux aspects du problème, à savoir l'aspect matériel et l'aspect humain. Les approches en question sont divisées en deux catégories, les approches séquentielles où la séquence de tâches est connue au départ et où les opérateurs sont affectés par la suite, et les approches simultanées où l'affectation des opérateurs et l'ordonnancement des tâches se font simultanément.

Nous avons étudié la complexité de nos problèmes dans les différents cas et développé des bornes inférieures pour l'évaluation des méthodes de résolutions que nous avons conçues. Ces dernières ont d'ailleurs donné de bons résultats, proches des bornes théoriques. Nous avons également pu fournir des méthodes de résolution de complexité polynomiale pour certains cas avec un et deux opérateur(s). Dans le cas de changement d'affectation en mode libre, nous

avons considéré l'hypothèse supplémentaire que les opérateurs avaient différents niveaux de performance. La modélisation proposée permet de représenter des opérateurs différemment de ce qui est fait dans la littérature. En effet, comme une machine a besoin d'un opérateur pour pouvoir exécuter des jobs, l'impact des opérateurs est modélisé par des temps de retard dus au fait que l'on a moins d'opérateurs que de machines. Nos études nous ont permis de conclure que l'ordre des jobs n'était pas si important pour un faible nombre d'opérateurs et qu'il est ainsi plus intéressant de considérer la gestion simultanée de ressources humaines et matérielles. Par contre, quand le nombre d'opérateurs est plus grand, les deux aspects du problème sont d'importance équivalente. Il est alors plus intéressant de résoudre le problème de manière séquentielle, à savoir construire un ordre des jobs, ensuite affecter au mieux les opérateurs. Les deux objectifs étudiés, à savoir la minimisation du temps total d'accomplissement ainsi que la minimisation du retard algébrique maximum sont importants en pratique. En effet, le premier assure une utilisation plus uniforme des machines et donc améliore l'efficacité du système de production. Le second trouve son utilité lorsqu'on a affaire à des dates d'échéance. Le respect de ces dernières ou du moins la minimisation des retards est un indicateur de fiabilité et de performance d'un système de production.

Nos résultats peuvent également servir de base à de futures études qui porteraient sur le développement d'autres bornes inférieures. De plus, les résultats de complexité permettraient de catégoriser des extensions ayant des paramètres additionnels tels que les temps de réglages et les dates de disponibilité.

Il serait aussi intéressant d'étudier d'autres objectifs comme la somme des temps de fin de traitement, qui est aussi d'une grand importance en pratique.

Dans le cas de changement d'affectation libre, comme le problème est $\mathcal{NP}$-difficile dans le cas où le nombre d'opérateurs est arbitraire, il serait intéressant de développer des méthodes

heuristiques pour sa résolution. Il serait aussi intéressant de tester les algorithmes polynomiaux sur des instances réelles, même lorsque $k > 2$. Par ailleurs, il y a lieu de considérer des contraintes supplémentaires permettant de rendre le modèle encore plus réaliste. Nous pourrions, par exemple, considérer les quarts de travail des opérateurs, les coûts occasionnés par l'emploi de ces derniers. Ceci ouvre la porte à toute une panoplie de nouvelles problématiques, toutes plus intéressantes les unes que les autres.

Pour finir, nous avons adapté certains de nos résultats aux ateliers de type job shop et open shop.

## BIBLIOGRAPHIE


Agnetis, A., M. Flamini, G. Nicosia, et A. Pacifici. 2011. « A job-shop problem with one additional resource type », *Journal of Scheduling*, vol. 14, p. 225–237.

Agnetis, A., G. Murgia, et S. Sbirilli. 2014. « A job shop scheduling problem with human operators in handicraft production », *International Journal of Production Research*, vol. 52 :13, p. 3820–3831.

Agrawal, S. 2011. « Minimizing the elapsed time with five machine flow shop scheduling and idle waiting time operator », *The IUP Journal of Operations Management*, vol. 10(3).

Allahverdi, A., C. Ng, T. Cheng, et M. Kovalyov. 2008. « A survey of scheduling problems with setup times or costs », *European Journal of Operational Research*, vol. 187, p. 985–1032.

Anand, E. et R. Panneerselvam. 2015. « Literature review of open shop scheduling problems », *Intelligent Information Management*, vol. 7, p. 33–52.

Baker, K. R., E. L. Lawler, J. K. Lenstra, et A. H. G. R. Kan. 1983. « Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints », *Operations Research*, vol. 31(2), p. 381–386.

Baki, M. 1999. « Some problems in one-operator scheduling ». Thèse de Doctorat, University of Waterloo.

Baptiste, P., V. Giard, A. Hait, et F. Soumis. 2005. *Gestion de production et ressources humaines*. Presses Internationales Polytechnique.

Baptiste, P. et A. Munier. 2013. « Ordonnancement sur machines parallèles avec partage d'opérateurs ». In *10ème Congrès International de Génie Industriel-CIGI 2013*.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Benavides, A., M. Ritt, et C. Miralles. 2014. « Flow shop scheduling with heterogenous workers », *European Journal of Operational Research*, vol. 237 (2), p. 713–720.

Benkalai, I., P. Baptiste, et D. Rebaine. 2015. « Ordonnancement d'ateliers de type flow shop avec contrainte d'opérateurs ». In *11ème Congrès International de Génie Industriel-CIGI 2015*.

———. 2016a. « Ordonnancement d'ateliers de type flow shop avec opérateurs en mode d'affectation libre ». In *17ème Conférence ROADEF de la Société Française de Recherche Opérationnelle et Aide à la Décision*.

Benkalai, I., D. Rebaine, et P. Baptiste. 2016b. « Assigning operators in a flow shop environment ». In *Information Systems, Logistics and Supply Chain*.

———. 2017a. « Ordonnancement d'ateliers de type job shop avec opérateurs ». In *18ème Conférence ROADEF de la Société Française de Recherche Opérationnelle et Aide à la Décision*.

———. 2017b. « Scheduling flow shops with operators », *International Journal of Production Research*, vol. DOI 10.1080/00207543.2018.1425557.

———. 2017c. « Scheduling job shops with operators in a free assignement changing mode ». In *7th International Conference on Industrial Engineering and Systems Management*.

———. 2018a. « Problème d'open shop avec opérateurs : nouveaux résultats de complexité ». In *19ème Conférence ROADEF de la Société Française de Recherche Opérationnelle et Aide à la Décision*.

———. 2018b. « Scheduling job shop problems with operators with respect to the maximum lateness (submitted) », *RAIRO-Operations Research*.

Benkalai, I., D. Rebaine, C. Gagné, et P. Baptiste. 2016c. « The migrating birds optimization metaheuristic for the permutation flow shop with sequence-dependent setup times ». In *8th IFAC conference on Manufacturing Modelling, Management and Control*.

———. 2017d. « Improving the migrating birds optimization metaheuristic for the permutation flow shop with sequence-dependent set-up times », *International Journal of Production Research*, vol. 55(20), p. 6145–6157.

Bierwirth, C. 1995. « A generalized permutation approach to jobshop scheduling with genetic algorithms », *Operations Research Spectrum*, vol. 17, p. 87–92.

Blazewicz, J., K. Ecker, E. Pesch, G. Schmidt, et J. Weglarz. 2001. *Scheduling computer and manufacturing processes*. Springer.

———. 2007. *Handbook on scheduling : From theory to applications*. Springer.

Blum, C. et A. Roli. 2008. « Hybrid metaheuristics : An introduction », *Studies in Computational Intelligence (SCI)*, p. 1–30.

Borreguero-Sanchidrian, T., R. Pulido, A. García-Sánchez, et M. Ortega-Mier. 2018. « Flexible job shop scheduling with operators in aeronautical manufacturing : A case study », *IEEE Access*, vol. 6, p. 224–233.

Brucker, P., J. Hurink, B. Jurisch, B. W. . Mashuda, et H. Ishii. 1997. « A branch and bound algorithm for the open shop problem », *Discrete Applied Mathematics*, vol. 76, p. 43–59.

Burns, F. et J. Rooker. 1976. « Johnson's three-machine flow-shop conjecture », *Operations Research*, vol. 24, p. 578–580.

Carniel, G., A. BEnavides, M. Ritt, et C. Miralles. 2015. « Including workers with disabilities in flow shop scheduling ». In *IEEE International Conference on Automation Science and Engineering (Vol. 2015-October, pp. 985–991). IEEE Computer Society*.

Cheng, T., G. Wang, et C. Sriskandarajah. 1999. « One-operator-two-machine flowshop scheduling with setup and dismounting times », *Computers & Operations Research*, vol. 26, p. 715–730.

Chernykh, I., A. Kononov, et S. Sevastyanov. 2013. « Efficient approximation algorithms for the routing open shop problem », *Computers and operations Research*, vol. 40, p. 841–847.

Cheurfa, M. 2005. « Gestion des ressources humaines en production cyclique ». Thèse de Doctorat, ENSM Saint-Étienne.

Cook, S. 1971. « The complexity of theorem proving procedures ». In *Proc. 3rd ACM Symposium on Theory of Computing*, p. 151–158.

Cook, W. 2012. *In pursuit of the Traveling Salesman*. Princeton University Press.

Dantzig, G., D. Fulkerson, et S. Johnson. 1954. « Solution of a large-scale Traveling Salesman Problem », *Operations Research*.

Dhiflaoui, M., H. Nouri, et O. Driss. 2018. « Dual-resource constraints in classical and flexible job shop problems : A state-of-the-art reviews », *Procedia Computer Science*, vol. 126, p. 1507–1515.

Dong, X., H. Huang, et P. Chen. 2009. « Study on heuristics for the permutation flowshop with sequence dependent setup times ». In *IEEE International Conference on Information Reuse & Integration*.

Duman, E., M. Uysal, et A. F. Alkaya. 2012. « Migrating birds optimization : A new meta-heuristic approach and its performance on quadratic assignment problem », *Information Sciences*, vol. 217, p. 65–77.

Fang, H., P. Ross, et D. Corne. 1994. « A promising hybrid ga/heuristic approach for open-shop scheduling problems ». In *11th European Conference on Artificial Intellignce*.

Farber, G., S. Salhi, et A. C. Moreno. 2007. « Sequencing in a non-permutation flow shop with constrained buffers : Applicability of genetic algorithm versus constraint logic programming ». In *XI Congreso de Ingenierìa de Organizaciòn International Conference on Industrial Engineering and Industrial Management Madrid*.

Fay, M. P. et M. A. Proschan. 2010. « Wilcoxon-mann-whitney or t-test ? on assumptions for hypothesis tests and multiple interpretations of decision rules », *Statistic Surveys*, vol. 4, p. 1–39.

Fister, I. J., X.-S. Yang, I. Fister, J. Brest, et D. Fister. 2013. « A brief review of nature-inspired algorithms for optimization », *ELEKTROTEHNISKI VESTNIK, English edition*, vol. 80(3), p. 1–7.

França, P., J. Gupta, A. Mendes, P. Moscato, et K. Veltink. 2005. « Evolutionary algorithms for scheduling a flow shop manufacturing cell with sequence dependent family setups », *Computers and Industrial Engineering*, vol. 48(3), p. 491 –506.

G. Campos Ciro, F. Dugardin, F. Y. R. K. 2014. « Open shop scheduling problem with a

multi-skills resource constraint : a genetic algorithm approach ». In *Meta 2014 the 5th international Conference on Metaheuristics and Nature Inspired Computing*.

Garey, M. et D. Johnson. 1979. *Computers and Intractability : A guide to the theory of NP-completeness*.

Giffler, B. et G. L. Thompson. 1960. « Algorithms for solving production-scheduling problems », *Operations Research*, vol. 8, p. 487.

Golden, B. L. et W. R. Stewart. 1985. « Empirical analysis of heuristics ». In *E. Lawler and J. K. Lenstra and A. H. G. Rinnooy Kan and D. B. Shmoys (eds) : The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization*. Wiley and Sons.

Graham, R. L. 1969. « Bounds on multiprocessor timing anomalies », *SIAM Journal of Applied Mathmatics*, vol. 17, p. 416–429.

Gueret, C. et C. Prins. 1999. « A new lower bound for the open shop problems », *Annals of Operations Research*, vol. 92, p. 165–183.

Gupta, J. 1986. « Flowshop schedules with sequence dependent setup times », *Journal of the Operational Research Society of Japan*, vol. 29 (3), p. 206– –219.

Gupta, J. et W. Darrow. 1986. « The two-machine sequence dependent flowshop scheduling problem », *European Journal of Operational Research*, vol. 24, p. 439–446.

Gupta, J. et E. Stafford. 2006. « Flowshop scheduling research after five decades », *European Journal of Operational Research*, vol. 169, p. 699–711.

Hall, N., H. Kamoun, et C. Sriskandarajah. 1997. « Scheduling in robotic cells : Classification, two and three machine cells », *Operations Research*, vol. 45(3), p. 421–439.

Hall, N., C. Potts, et C. Sriskandarajah. 2000. « Parallel machine scheduling with a common server », *Discrete Applied Mathematics*, vol. 102, p. 223–243.

Hendizadeh, S., H. Faramarzi, S. Mansouri, J. Gupta, et T. ElMekkawy. 2008. « Metaheuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times », *International Journal of Production Economics*, vol. 111(2), p. 593– 605.

Horvath, E. C., S. Lam, et R. Sethi. 1977. « A level algorithm for preemptive scheduling », *J. Assoc. Copmut. Mach.*, vol. 24(1), p. 32–43.

Johnson, S. 1954. « Optimal two and three-stage production schedules with setup times included », *Naval research logistics quarterly*, vol. 1(1), p. 61–68.

Jurisch, B. et W. Kubiak. 1997. « Two-machine open shops with renewable resources », *Operations Research*, vol. 45, p. 544–552.

Karp, R. 1972. *Complexity of Computer Computations*, p. 85–103. Plenum Press,New York.

Kis, T., D. de Werra, et W. Kubiak. 2010. « A projective algorithm for preemptive open shop scheduling with two multiprocessor groups », *Operations Research Letters*, vol. 38, p. 129–132.

Koulamas, C. 1998. « A new constructive heuristic for the flowshop scheduling problem », *European Journal of Operational Research*, vol. 105, p. 66–71.

Kumar, G. et S. Singhal. 2013. « Genetic algorithm optimization of flow shop scheduling problem with sequence dependent setup time and lot splitting », *International Journal of Engineering, Business and Enterprise Applications*, vol. 4(1), p. 62–71.

Lawler, E. L. 1982. *Preemptive scheduling of precedence-constrained jobs on parellel machines*. Coll. « Deterministic and stochastic scheduling, Proceedings of the NATO

advanced study and reasearch institute on theoretical approaches to scheduling problems », p. 101–123. D. Reidel publishing Co.

Lee, I.-S. 1991. « A worst-case performance of the shortest-processing-time heuristic for single machine scheduling », *The Journal of the Operational Research Society*, vol. 42(10), p. 895–901.

Li, X., M. F. Baki, et Y. P. Aneja. 1995. « Flow shop scheduling to minimize the total completion time with a permanently present operator : Models and ant colony optimization metaheuristic », *Computers and Operations Research*, vol. 38, p. 152–164.

Liao, C. J., L. M. Liao, et C. T. Tseng. 2007. « A performance evaluation of permutation vs. non-permutation schedules in a flowshop », *International Journal of Production Research*, vol. 44 (20), p. 4297–4309.

Liaw, C. 1999. « A tabu search algorithm for the open shop scheduling problem with sequence dependent setup times », *Computers and Operations Research*, vol. 26, p. 109–126.

Lin, S.-W., K.-C. Ying, C.-C. Lu, et J. Gupta. 2011. « Applying multi-start simulated annealing to schedule a flowline manufacturing cell with sequence dependent family setup times », *International Journal of Production Economics*, vol. 130, p. 246– 254.

Liu, C. et R. Bulfin. 1988. « Scheduling open shops with unit execution times to minimize functions of due dates », *Operations Research*, vol. 36, p. 553–559.

Lu, L. et M. E. Posner. 1993. « An np-hard open shop problem with polynomial avarage timpe complexity », *Mathematics of Operations Research*, vol. 18, p. 12–38.

Mashuda, T. et H. Ishii. 1994. « Two machine open shop scheduling problem with bi-criteria », *Discrete Applied Mathematics*, vol. 52, p. 253–259.

Matta, M. 2009. « A genetic algorithm for the proportionnate multiprocessor open shop », *Computers and Operations Research*, vol. 36, p. 109–126.

Mencia, C., M. R. Sierra, M. A. Salido, J. Escamilla, et R. Varela. 2015a. « Solving the job shop scheduling problem with operators by depth-first heuristic search enhanced with global pruning rules », *Artificial Intelligence Communications*, vol. 28, p. 365–381.

Mencia, C., M. R. Sierra, et R. Varela. 2013. « An efficient hybrid search algorithm for job shop scheduling with operators », *International Journal of Production Research*, vol. 51 :17, p. 5221–5237.

Mencia, R., M. R. Sierra, C. Mencia, et R. Varela. 2011. *Genetic Algorithm for job shop with operators*. Coll. « Ferrandez J. M., Alvarez Sanchez J. R., de la Paz F., Toledo F. J. (eds.) New challenges on bioinspired applications. IWINAC. Lecture Notes in Computer Science ». T. 6687. Springer, Berlin, Heidelberg.

———. 2014. « A genetic algorithm for job shop scheduling with operators enhanced by weak lamarckian evolution and search space narrowing », *Natural Computing*, vol. 13, p. 179–192.

———. 2015b. « Memetic algorithms for the job shop scheduling problem with operators », *Applied soft computing*, vol. 34, p. 94–105.

Morinaga, E., Y. Sakaguchi, H. Wakamatsu, et E. Arai. 2017. « A method for flexible job-shop scheduling considering workers and teams ». In *9th International Conference on Leading Edge Manufacturing in 21st Century, LEM 2017*.

Nawaz, M., E. J. Enscore, et I. Ham. 1983. « A heuristic algorithm for the m-machine n-job flow-shop sequencing problem », *Omega. The International Journal of Management Science*, vol. 11, p. 91–95.

Neubert, G. et M. Savino. 2009. « Flow shop operator scheduling through constraint satisfaction and constraint optimisation techniques », *International Journal of Productivity and Quality Management*, vol. 4(5/6).

NoorulHaq, A., M. Saravanan, A. Vivekraj, et T. Prasad. 2007. « A scatter search approach for general flowshop scheduling problem », *International Journal of Advanced Manufacturing Technology*, vol. 31, p. 731–736.

Or, I. 1976. « Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking ». Thèse de Doctorat, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.

Oulamara, A., D. Rebaine, et M. Serairi. 2013. « Scheduling the two-machine open shop problem under ressource constraints for setting the jobs », *Annals of Operations Research*, vol. 211, p. 333–356.

Paksi, A. et A. Ma'Ruf. 2016. « Flexible job-shop scheduling with dual-resource constraints to minimize tardiness using genetic algorithm ». In *IOP Conference Series : Materials Science and Engineering (Vol. 114). Institute of Physics Publishing*.

Panahi, H. et R. Tavakkoli-Moghaddam. 1994. « Solving a multi-objective open shop scheduling problem by a novel hybrid ant colony optimization », *Expert systems with applications*, vol. 38, p. 2817–2822.

Paschos, V. 2005. *Optimisation combinatoire 1, concepts fondamentaux*. Hermès science.

Pinedo, M. 2002. *Scheduling : Theory, algorithms and systems*. Prentice Hall.

Potts, C., D. Shmoys, et D. Williamson. 1991. « Permutation vs. non-permutation flowshop schedules », *Operations Research Letters*, vol. 10, p. 281–284.

Pérez-González, P., J. M. F. Torres, P. L. Gonzalez-R, J. M. L. Blanco, et R. R. Usano. 2009. « Flowshop scheduling problems with due date related objectives : A review of the literature ». In *3rd International Conference on Industrial Engineering and Industrial Management XIII Congreso de Ingenieria de Organizacion, Barcelona-Terrassa, September 2nd-4th*.

Rìos-Mercado, R. et J. Bard. 1998. « Heuristics for the flow line problem with setup costs », *European Journal of Operational Research*, vol. 110, p. 76–98.

Roshanaei, V., M. Esfehani, et M. Zandieh. 2010. « Integrating non-preemptive open shop scheduling with sequence dependent setup times using advanced metaheuristics », *Expert systems with applications*, vol. 37, p. 259–266.

Ruiz, R., M. Concepción, et J. Alcaraz. 2004. « Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics », *European Journal of Operational Research*, vol. 217, p. 34–54.

Ruiz, R. et T. Stutzle. 2005. « An iterated greedy algorithm for the flowshop problem with sequence dependent setup times ». In *MIC2005. The 6th Metaheuristics International Conference*.

Seidgar, H., M. Abedi, et S. Rad. 2015. « A new mathematical model for scheduling flexible flow shop problem with learning and forgetting effects of workers », *International Journal of Industrial and Systems Engineering*, vol. 21(4), p. 534–549.

Sha, D., H. Lin, et C. Hsu. 2010. « A modified particle swarm optimization for multi-objective open shop scheduling ». In *proceedings of the international multi conference of engineers and computer scientists*.

Sierra, M. R., C. Mencia, et R. Varela. 2011. *Optimally scheduling a job shop with operators and total flow time*. Coll. « Lozana J. A., Ganaz J. A., Moreno J. A. (eds.) Advances in Artificial intelligence. CAEPIA. Lecture Notes in Computer Science ». T. 7027. Springer, Berlin, Heidelberg.

———. 2015. « New schedule generation schemes for the job shop problem with operators », *Journal of intelligent manufacturing*, vol. 26, p. 511–525.

Talbi, E.-G. 2009. *Metaheuristics, from design to implementation*. John Wiley & Sons, Inc.

Tongur, V. et E. Ulker. 2014. « Migrating birds optimization for flow shop sequencing problem », *Journal of Computer and Communications*, vol. 2, p. 142–147.

Ullman, J. D. 1976. *Complexity of sequencing problems*. Coll. « Computer and Job/shop scheduling theory ». Wiley & Sons, Inc. New York.

Vahedi-Nouri, B., P. Fattahi, et R. Ramezanian. 2013. « Minimizing total flow time for the non-permutation flow shop scheduling problem with learning effects and availability constraints », *Journal of Manufacturing Systems*, vol. 32, p. 167–173.

Vallada, E., R. Ruiz, et C. Maroto. 2003. Synthetic and real benchmarks for complex flow-shop problems. Rapport, Universidad Politecnica de Valencia, Valencia, Espana, Grupo de Investigation Operativa GIO.

Vickson, R. 1980a. « Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine », *Operations Research*, vol. 28(5), p. 1155–1167.

———. 1980b. « Two single machine sequencing problems involving controllable job processing times », *AIIE transactions*, vol. 12(3), p. 258–262.

Ying, K.-C., J. Gupta, S.-W. Lin, et Z.-J. Lee. 2010. « Permutation and non-permutation schedules for the flowline manufacturing cell with sequence dependent family setups », *International Journal of Production Research*, vol. 48(8), p. 2169–2184.

Ying, K.-C. et S.-W. Lin. 2007. « Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems », *International Journal of Advanced Manufacturing Technology*, vol. 33, p. 793–802.

Ziaee, M. et S. Sadjadi. 2007. « Mixed binary integer programming formulations for the flow shop scheduling problems. a case study : ISD projects scheduling », *Applied Mathematics and Computation*, vol. 185, p. 218–228.

Zouba, M. 2009. « Ordonnancement de machines parallèles identiques avec des contraintes de ressources humaines ». Thèse de Doctorat, École Polytechnique Montréal.